AFRL-IF-RS-TR-2002-73
Final Technical Report
April 2002

APPLICATION OF MODEL-BASED REASONING TOOLS USED TO ENHANCE AND IMPROVE DIAGNOSTIC PERFORMANCE TO IMPROVE AIR FORCE MAINTENANCE

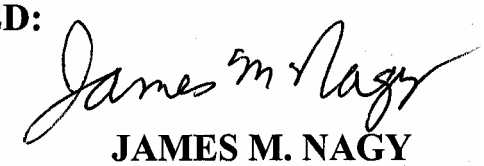
Giordano Automation Corporation

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-73 has been reviewed and is approved for publication.

APPROVED:



JAMES M. NAGY
Project Engineer

FOR THE DIRECTOR:

MICHAEL L. TALBERT, Technical Advisor
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Apr 02	Final Aug 99 – Dec 01

4. TITLE AND SUBTITLE
APPLICATION OF MODEL-BASED REASONING TOOLS USED TO ENHANCE AND IMPROVE DIAGNOSTIC PERFORMANCE TO IMPROVE AIR FORCE MAINTENANCE

5. FUNDING NUMBERS
C - F30602-99-C-0175
PE - N/A
PR - TEMS
TA - 01
WU - 01

6. AUTHOR(S)
Mary Nolan, Gerard Giordano, Brian Gaboda, Al Esser, Giordano Automation Corporation

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Giordano Automation Corporation
21 White Deer Plaza
Sparta, NJ 97871

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
LEADA
Warner Robins AFB, GA 31098

AFRL/IFTD
525 Brooks Road
Rome NY 13441-4514

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2002-73

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: James M. Nagy, IFTD, 315-330-3173, nagyj@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

The major objective of this effort was to provide enhancements to the maintenance of the A-10/KC-135 Turbine Engine Monitoring System (TEMS) through implementing the Diagnostician model-based reasoning tool in a selection of Shop Replaceable Units (SRU) level test program sets. This effort included re-engineering of the TEMS SRU level test programs to improve run-time efficiency, accuracy and vertical testability. The TEMS performs parametric analysis of KC-135 and A-10 engine data. The TEMS unit is mounted on the aircraft. The TEMS LRU and SRU level testing is performed at the Warner Robins Air Logistics Center (WRALC) in Georgia (previously at Kelly AFB in San Antonio). Since the SRU and LRU level test resources are co-located at the same facility, a rare opportunity exists to analyze the level of test result consistency across the two testers.

14. SUBJECT TERMS
Diagnostics, Test Program Set Development, Maintenance, Model-Based Reasoning

15. NUMBER OF PAGES
53

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Tables

List of Illustrations

Appendix C Illustrations

Program Final Report

1.0 INTRODUCTION

The major objective of this effort was to provide enhancements to the maintenance of the A-10/KC-135 Turbine Engine Monitoring System (TEMS) through implementing the Diagnostician model-based reasoning tool in a selection of Shop Replaceable Units (SRU) level test program sets. This effort included re-engineering of the TEMS SRU level test programs to improve run-time efficiency, accuracy and vertical testability.

1.1 Background

The Turbine Engine Monitoring System (TEMS) performs parametric analysis of KC-135 and A-10 engine data. The TEMS unit is mounted on the aircraft. The TEMS LRU and SRU level testing is performed at the Warner Robins Air Logistics Center WRALC in Georgia (previously at Kelly AFB in San Antonio). Since the SRU and LRU level test resources are co-located at the same facility, a rare opportunity exists to analyze the level of test result consistency across the two testers.

In the past few years, the TEMS SRU level-testing software was re-hosted from a VAX controller to a PC controller. At that time, significant inefficiencies in the structure and documentation of the test programs were identified. Inconsistencies were also identified between the LRU and SRU level tests. A proof-of-concept demonstration was conducted in which it was determined that by applying reasoning tools to the test programs, that the run-time speed and test program accuracy were significantly enhanced. At the same time, the structured, engineering process required to implement the Diagnostician resulted in identification of specific problem areas, which could then be resolved. The proof–of-concept demonstration was performed on 091350 RPM Fuel Flow Conditioner circuit card. Table 1 below shows the results of the demonstration.

Table 1 - Diagnostician Benefits to TEMS SRU TPS

Demonstration done on 091350 TEMS A6 Card

Run #	Fault Inserted	Original Time	Diagnostician Time	Time Saved	% Faster	Original Callout	Diagnostician Callout
Run 1	Go-chain	00:23:51	00:15:03	00:08:48	36.9%	Pass	Pass
Run 2	U5.3 SA0	00:22:57	00:06:53	00:16:04	70.0%	U5,U13,U27, U12	U5,U2,U13,U27 Jumper
Run 3	U8.11 SA0	00:13:48	00:05:17	00:08:31	61.7%	AR2,C2,R3,R4	U8,R7,R8
Run 4	U7.13 SA0	00:23:11	00:04:26	00:18:45	80.9%	AR2,C2,R3,R4	U7
Run 5	U28.4 SA0	00:18:45	00:06:23	00:12:22	65.9%	U16,U17,U18,U19 U20,U21,U28	U16,U17,U18,U19 U20,U21,U28
Run 6	AR1.10 SA0	00:16:20	00:04:55	00:11:25	69.9%	AR1,R1,R2	AR1, R1, R2
Run 7	U7.11 SA0	00:15:30	00:06:07	00:09:23	60.5%	AR2,C2,R3,R4	U7,R7,R8
Run 8	AR1.3 SA0	00:16:04	00:06:53	00:09:11	57.2%	AR1,R1,R2	AR1, R1,R2
Run 9	U12.2 SA0	00:20:41	00:08:02	00:12:39	61.2%	U12,U3,U16	U12,U3,U16

The demonstration also determined that the commonality of test results between the SRU and LRU level tests could be increased and the mechanism could be put in place to upgrade the re-engineered test programs based on test results and correlation over time and history. Many of the various TPS improvements came from the run-time characteristics of the reasoning tools as well as the application of

the structured engineering process for development of a proper diagnostic knowledge base for use with the Diagnostician. The result of re-engineering the TPSs on the rest of the TEMS cards as part of this project likewise verified the significant improvement in TPS quality in terms of both the Go-chain and the Diagnostic process. These quality improvements are detailed in Section 2.

The efforts described in this Final Report were based upon a contract to apply the Diagnostician and the engineering analysis across all of the TEMS EPU circuit cards. The automatic diagnostic reasoning approach that Giordano Automation used in re-engineering the test program sets has been accomplished using a set of tools developed by Giordano Automation. The run-time tool, called the Diagnostician, provides automated diagnostics that is integrated into the Test Program. The development tool, the Diagnostic Profiler was used to create the Diagnostic models. The tools are summarized in section 1.4. A more detailed description is provided in Appendix C.

1.2 Test Station Environment

The Mate 390 System is an existing Test Station located at Warner Robins ALC. It was built in the late 1980's. It was upgraded from its original MicroVAX computer controller configuration to a PC-based controller and a SCO UNIX operating system a few years ago. The Air Force standard in Test System architecture throughout the 1980's was defined in the Modular Automatic Test System (MATE) standards. The MATE system was a system of standards aimed at increasing the commonality across test systems. The MATE 390 station conforms to these standards. The MATE program included standard Control and Support Software that was used in each test system application.

1.3 MATE Control and Support Software

When the system was upgraded from the test station MicroVAX controller to the PC controller, a similar operating system environment was hosted on the PC. The Operating System selected was UNIX by Santa Cruz Operations (SCO). At the time, the SCO Unix offered a convenient solution to porting software from the VAX operating system, which was Unix-based, to the PC. The SCO UNIX operating system was hosted on the PC controller. A C compiler based upon the commonly available GNU software was compiled. This enabled transition of the MATE control and support software to the SCO UNIX environment.

1.4 Diagnostic Profiler And The Diagnostician

Giordano Automation has developed a powerful set of tools that implement model-based diagnostic reasoning. The run-time tool, Diagnostician, provides automated diagnostics and can be seamlessly integrated into any test environment. The development tool, the Diagnostic Profiler, assists the engineer in developing the run-time diagnostic knowledge base. The Diagnostician is an implementation of model-based reasoning. Model-based reasoning means that a diagnostic model of a system or item, derived from design data, serves as the basis for diagnostic reasoning. The diagnostic model is independent of the test program and independent of the sequence of tests that are run.

The model-based diagnostic software object called the Diagnostician was used in lieu of programmed fault trees. In run-time, the Diagnostician provides dynamic fault isolation without complex diagnostic logic paths, by reading test results. The diagnostic logic is not "fixed" to a pre-determined, static diagnostic tree, but rather is dynamic. The Diagnostician dynamically interprets test results - test

results can come from any source, in any order, and with as many or as few test results at a time as the test source can provide. Static test trees, on the other hand, are based upon one test result at a time, in a pre-determined sequence, and from a fixed test source.

The Diagnostician contains a diagnostic model of the item automatically converted from design data. The model is in the form of a connectivity matrix that represents the propagation of faults (rows in the matrix) to observable measurement locations and the coverage of tests that Pass or Fail (columns in the matrix). When used in run-time, the software algorithms and knowledge base (matrix) operate to isolate faults without hard-coded diagnostic test sequences.

In run-time, the Diagnostician interprets, in real time, test results to perform fault isolation. The concept of object-oriented programs is taken full advantage of by dealing with the diagnostic logic as an independent entity of the test program. By separating the diagnostic logic from the test, the test program becomes significantly simpler. Further, the diagnostic logic contained in the software object can be rehosted to any platform without any problem, because it is simply a binary file.

Using the Diagnostician, the fundamental culture of diagnostics has been changed. Tests perform measurements and data collection and determine if those measurements are within acceptable ranges. The interpretation of what it means if the measurement has passed or failed is done by the Diagnostician, which dynamically, on-the-fly, interprets test information based upon all information it receives in any order.

The Diagnostician makes use of "Minimum Set Covering" algorithms that interpret the "Cones of Evidence" produced by both pass and fail test result data. These reasoning techniques provide for fast, accurate, flexible diagnostics, and can also isolate multiple faults. Static test trees, on the other hand, are limited to a "single fault assumption" and often do not work in a multiple fault situation.

The Diagnostic Profiler supports the development of the diagnostic software object via a diagnostic model. The selection of test points and the assessment of fault isolation probabilities as well as validation of these probabilities are all done using the Diagnostic Profiler during development of the TPS. Diagnostic engineering and test engineering are uncoupled. Test programming tools are used to write tests. In the process of writing these tests, the test engineer defines the Pass/Fail (P/F) criteria for each response value being measured and converts test result data for each measured parameter into a P (Pass) or F (Fail). This function can be implemented utilizing a simple high level language subroutine that accepts measurement test results and associated tolerances values as inputs and outputs a "P/F" character.

Use of the diagnostic object in run-time to perform fault isolation is done by the Diagnostician. To incorporate diagnostics into the test program, a single "WHILE" loop is incorporated into the Test Program, in this case ATLAS. If there is another test that can further isolate the fault, the run time directs the Diagnostician for the next optimum test to perform, runs that test, and sends test results to the Diagnostician.

Refer to Appendix C for an in-depth discussion of the Diagnostic process used in this project.

2.0 SUMMARY OF PROJECT RESULTS

The complexity of the TEMS TPS code has been significantly simplified by inserting the Diagnostician. The traditional troubleshooting trees that were previously implemented with several, hard to maintain GOTO statements, were replaced with a simple conversation loop with the Diagnostician. By eliminating this complex diagnostic hard-coded logic, the resulting TPSs are vastly easier to maintain. Also, transporting the modified TPSs and the Diagnostician to an alternate test resource is much more straightforward. This approach has also allowed for a significant reduction in the number of lines of code for each Test Program as is shown in Table 2.

TABLE 2
TPS PROJECT COMPLETION SUMMARY

TPS	Old TPS # Lines	New TPS # Lines	# Old Probes	# New Probes	Go-To's Removed	Modifications Compared to old code
091150	16,468	10,770	58	18	162	Significantly reduced # of probes and code lines
091200	9,715	9,300	76	10	221	Significantly reduced # of probes R1 test was added
091250	10,524	7,492	51	10	219	Significantly reduced # of probes
091300	4,521	6,482	41	28	51	- WB Diag test added to Go-chain to reduce ambiguity - Runs R76 & R4 first to reduce ambiguity - WB, NB & VIBCLK tests results are displayed in log files as applied - tolerances were tightened accordingly
091350	28,524	11,532	117	50	1401	Reduced # of probes
091450	combined w/ 091460	-			combined w/ 091460	- combined 091450 & 091460 TPS's to one linked ATLAS program
091460	16,553	24,551	57	29	732	Added a calibration test to the potentiometer on the 4.9 Volt Reference Test.
9383755	N/A	12,436	N/A	28	N/A	New Program.
091600	78,589	14,099	113	72	1422	Combined all 4 old mod code into 1 linked TPS program
091650	55,493	8,851	69	30	1804	Combined all 6 modules into 1 linked program
091750	19,977	-	86	-	-	Pushed to PRDA-2

The overall test results have been significant in that we see a vast improvement in the overall diagnostics, a reduction in the amount of probes in general on each individual board, and the re-orientation and modular structuring of the test program to allow it to be easily migrated to another functional test system. In addition, the Diagnostic Profiler can be applied directly to those comparable tests on any migrated test system to allow capturing of the diagnostic data as you migrate from one tester to the other. This would be a significant reduction in overall test program costs in migration of the test programs to alternate functional test system. Appendix A contains a listing of all the Software Delivery

Forms for the various assemblies that have been certified. These items have been certified through the LYSTA organization of the Warner Robins Air Logistic Center (WRALC) Software Development department and transmitted to the TEMS Equipment Specialist for displacement and disposition for use on the Mate 390 Test System.

In general the test programs have been dramatically improved on the go chain to increase accuracy where correlation problems have existed between the LRU and the SRU test system. A major improvement was to separate the various test program sub-sections into modular stand-alone tests, which can be easily maintained, de-bugged and transported. In addition, all documentation and supporting information is provided to the Air Force as part of this contract in order to allow total organic maintenance and support of these Test Programs in the future. Due to the use of the Diagnostician, a more accurate and efficient resolution to the specific component failure is realized with the upgraded diagnostics. The diagnostic process for the boards all exhibit a reduction of the number of probes from the previous test programs in order to accomplish an improved diagnostic environment. In addition, the more complicated "fault tree" approach to diagnostics has been eliminated. In effect, a model has replaced the manual fault tree depiction of the individual probe processes. The diagnostic model is much easier to maintain and upgrade and support as any discrepancies or anomalies occur. A major benefit is that the diagnostic process through the Diagnostician allows a direct application and migration to a migrated test program on another Test platform as the MATE 390 is phased out in the future.

3.0 TASKS AND TECHNICAL REQUIREMENTS.

Under this contract, the following tasks and requirements were defined and accomplished:

3.1 Re-Engineer and Incorporate Advanced Diagnostics in the TEMS SRU Test Program Sets

The major task performed on this contract was the re-engineering of the TEMS SRU Test Program Sets (TPSs) on the MATE 390 test system to incorporate the reasoning tool (Diagnostician) to perform model-based diagnostics. Table 3 shows the applicable SRU TPSs, which were re-engineered under this contract.

TABLE 3
TEMS SRU Test Program Sets

CPIN	*SRU*	*TEMS EPU Slot Configuration*
85E-USQSS/M390-U013-00A	**091150**	**A2**
85E-USQ85/M390-U004-00A	**091200-301,302**	**A3**
85E-USQ85/M390-U005-00A	**091250-302**	**A4**
85E-USQ85/M390-U006-00A	**091300-303,302**	**A5**
85E-USQ85/M390-U007-00A	**091350-302,304,305,306**	**A6**
85E-USQS5/M390-U014-00A	**091450-(301-314)**	**A8**
85E-USQ85/M390-U014-00A	***091 460-(301-306)**	**A8**
85E-USQ85/M390-U014-00A	**9383755-10**	**A8**
85E-USQ85/M390-U008-00A	**091600 –301 thru –308, 311 thru -318, 322, 323, 325, 326**	**A11**
85E-USQ85/M390-U009-00A	**091650-303,304 (six configurations)**	**A10, A12**
85E-USQ85/M390-U011-00A	****091750-301**	**A13**

* This card is very similar to the 091450. The models and programs are similar enough that one model and one test program can be used for all variations/revisions of the 091450 and 091460 A/D converter cards respectively.

** This is an assembly of 2 CCA's and a Filter

3.2 Sequence of Tasks to Re-engineer the TEMS SRU Test Programs

For each of the SRUs, the following tasks have been performed to implement the reasoning tool, the Diagnostician:

1. Model the Unit Under Test (UUT) in the OrCAD schematic capture CAD tool. Correlate the schematics with actual hardware to incorporate corrections into the schematics. Import the CAD model netlist representation (in EDIF format) into the diagnostics development tool, the Diagnostic Profiler.
2. Review the test program code. Identify portions of test code that are inefficient or have errors. Correct all identified errors and streamline test code per findings.
3. Perform UUT failure analysis of TPS tests versus fault coverage, using the development tool, the Diagnostic Profiler.
4. Verify and validate the resultant Diagnostic Knowledge Base
5. Integrate the TPS with the diagnostic model.
6. Verify the completed TPS on MATE 390 and prepare a data package.
7. Perform a sell-off of each TPS to the designated WRALC Air Force software representatives including fault insertions.
8. Provide updated CPIN software on suitable media for release and distribution.
9. Update TPI (Test Program Instruction) documentation

Step 1 – Model the UUT in OrCAD

Not only does the Diagnostician achieve fast, accurate diagnostics, but also the process associated with the implementation of the Diagnostician results in dramatic TPS improvements. This section will provide an example of this process using a portion of one of the TEMS SRUs. A portion of the SRU is used to increase the understandability of this sample. The 091200 card, which is the Bridge, Temperature and Switch Conditioner circuit card, will be used for this example.

The first step is to model the Unit Under Test in the OrCAD schematic capture tool. Within this step, the schematic diagrams from the Air Force Tech Orders and the Original Equipment Manufacturers are reviewed and compared against the actual hardware. In many cases, errors have been found in the schematics contained in the Air Force Tech Orders. Additionally, schematics often do not represent the large number of circuit card versions and revisions that have been performed over the twenty years since the original design of the TEMS system. Performing this analysis enables the converging on a schematic representation that is accurate, correct, and that accounts for all allowable revisions and versions of the board, as appropriate.

Once input to OrCAD (or any other CAD system), an EDIF (Electronic Design Interchange Format) netlist file can be generated as a file format output from the CAD system. EDIF is an Industry standard, IEEE format for definition of electronic designs. EDIF netlist information includes part definition, interconnectivity, and signal flow.

The EDIF netlist file is used by the Diagnostic Profiler “Import Design” tool to capture design information and create a diagnostic representation of the design depicting signal flow, fault propagation and test accessibility to the internal portions of the design. The diagnostic model that results is a “fault/symptom matrix” that is the basic information format used by the Diagnostic Profiler.

Figure 1 below shows the portion of the 091200 card schematic diagram from OrCAD.

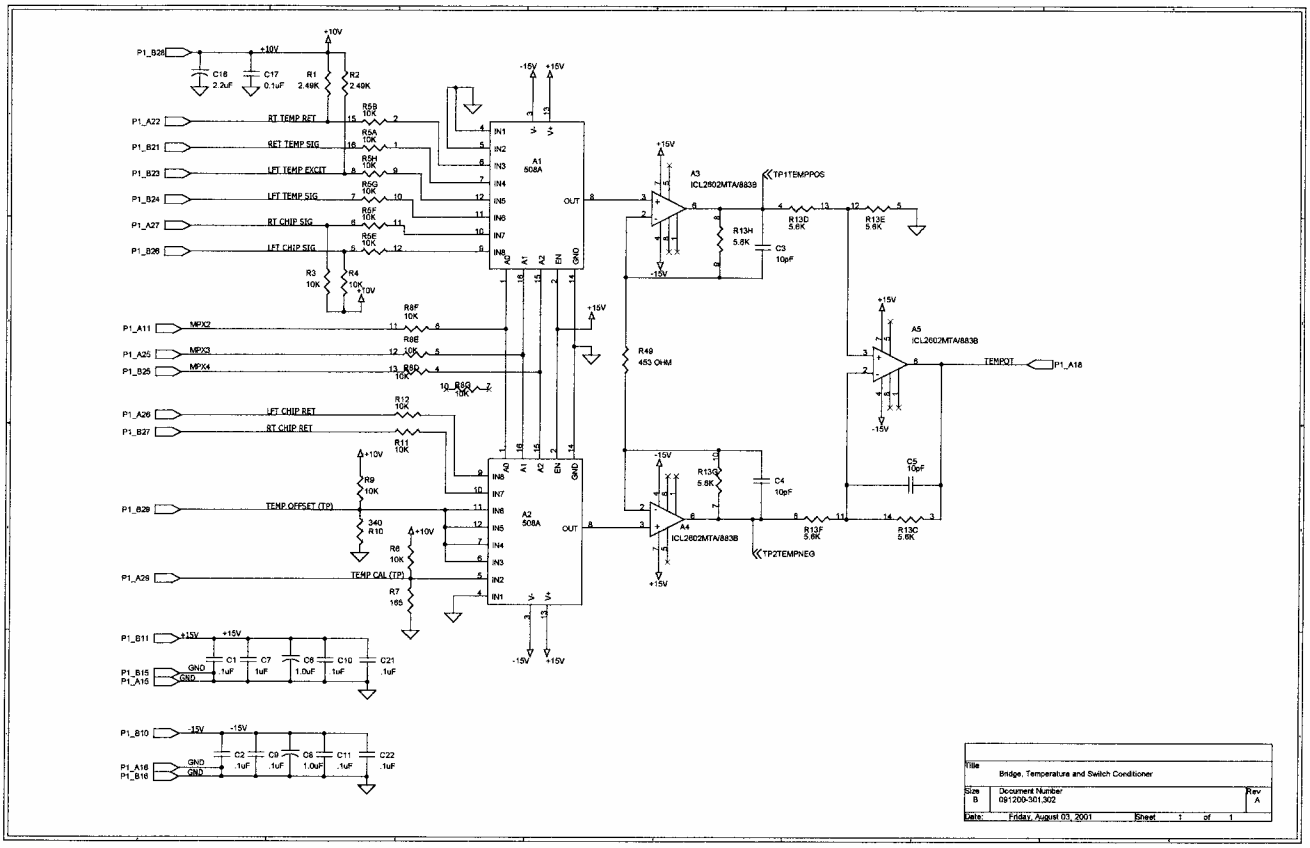


Figure 1 – 091200 Schematic Representation

Figure 2 shows a selected screen image of the design import tool that imports the EDIF netlist into the Diagnostic Profiler to create a diagnostic model.

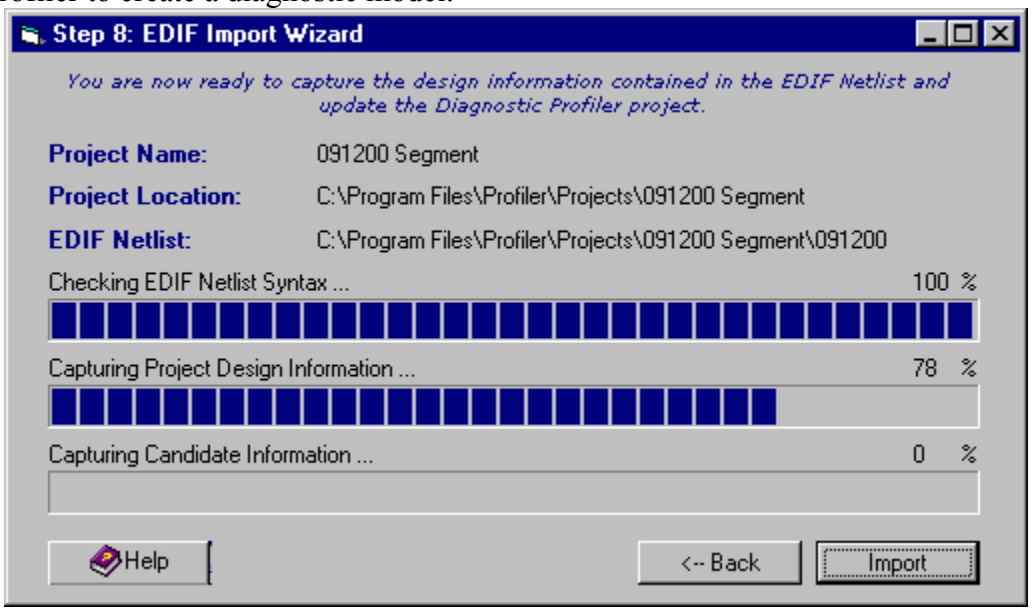


Figure 2 – Diagnostic Profiler EDIF Import Tool

Step 2 – Test Code Review and Streamlining

The second step in the process is to analyze the test code. Portions of the test code that are inefficient or have errors are identified. All identified errors are corrected and the test code is streamlined per the findings of the analysis.

Referring back to Table 2, the full 091200 circuit card test program originally contained 9715 lines of ATLAS code. The streamlining of test code and the integration of the Diagnostician resulted in a reduction in the number of lines of code down to 9300 lines. In the original code, there were 76 probe routines. These were reduced down to ten (10) significant probe routines. Additionally, a test that was diagnostically relevant, but not included in the original test program (test of R1) was added.

A typical ATLAS test program is made up of numerous conditional branches leading to ATLAS "GO TO" statements. The GO TO statements handle the program's traversal through diagnostic logic to lead to a fault call-out. The 091200 card contained 221 individual Go To statements related to diagnostic logic. These 221 Go To statements were completely eliminated in the ATLAS test program because the diagnostic logic is contained within the Diagnostic Knowledge Base, and the Diagnostician, in run-time, manages all test sequencing and traversal through test routines to achieve the fault call-out.

This results in significantly easier to maintain test programs and a test program which is more efficient in run-time. The resulting test program is also much easier to understand.

The diagnostic logic contained in just one single diagnostic path in the old ATLAS code testing the 091200 circuit card is shown below. By using the Diagnostician, this code as well as rest of the diagnostic logic fault tree code has been eliminated from the 091200 TPS. It is intuitively obvious that following the code path and maintaining the code for even a signal path in the hard-coded fault tree logic is complicated. Multiply this by the many fault tree paths contained in any test program, and benefits become very clear of the increased maintainability and portability of the new TPS code where all this logic is replaced by a single diagnostic loop and a diagnostic model.

```
134000 SETUP, DIGITAL TEST, TYPE PARALLEL,
          VOLTAGE-ONE   15.0 V,
          VOLTAGE-ZERO -15.0 V,
          CNX-STIM HI P1-B25 P1-A25 P1-A11 $

B       BRANCH FROM STEP 133350 $

    10 FILL, 'TST NAME',
          C'TEMPOT VDC OUTPUT TEST 1340' $

    20 OUTPUT,
          EXECUTING ('TST NAME') $

    30 DO, DIGITAL TEST, STIM-ONLY,
          STIM X'2',
          WORD-RATE 100.0 WORDS/SEC,
          CNX-STIM HI P1-B25 P1-A25 P1-A11 $

C       CONNECT P1-A22 TO 0.627 VDC FROM
RESISTOR NETWORK
       FORMED BY R1 IN UUT AND 166.5 OHM IN ITA
$

    40      CONNECT, SHORT,
              CNX FROM XA6-51 TO P1-A22 $

C       A1-8=0.627 VDC, A2-8=0.329 VDC
        VERIFIE OUTPUT AT P1-A18 IS 7.664 +/-0.514 $

    50 FILL, 'TYPE', 'TST NMBR', 'DIMEN',
             1,     C'134000',  C'VDC' $

    60 FILL, 'PIN HI',   'PIN LO',  'UPR LMT',  'LWR
LMT',
             C'P1-A18',  C'SYSGND',  8.178,      7.150 $

    70 VERIFY, (VOLTAGE INTO 'MSRMNT'),
          DC SIGNAL,
          UL 'UPR LMT' V LL 'LWR LMT' V,
          VOLTAGE MAX 15.0 V,
          CNX HI P1-A18 LO DMMLO6 $

    80 IF, NOGO, THEN $
```

```
    90      PERFORM, 'FAILR MSG' $

 134100     GO TO, STEP 707000 $

    10 END, IF $
```

… Note: Other Code Mixed in Here That is Not Relative to This Diagnostic Path

```
707000 FILL, 'TST NMBR', 'UPR LMT', 'LWR LMT',
            C'707010',  -14.0,      -16.0 $

    10 PERFORM, 'A1-1 INPUT' $

C       FILL LIMITS FOR TEST POINT A1-16 $

    20 FILL, 'TST NMBR', 'UPR LMT', 'LWR LMT',
            C'707030',   16.0,      14.0 $

    30 PERFORM, 'A1-16 INPUT' $

C       FILL LIMITS FOR TEST POINT A1-15 $

    40 FILL, 'TST NMBR', 'UPR LMT', 'LWR LMT',
            C'705050',  -14.0,     -16.0 $

    50 PERFORM, 'A1-15 INPUT' $


C       ***********************************
        *  TEST A1-8 USING ANALOG PROBE   *
        ***********************************$

C       TEST PASSES IF MEASURED VOLTAGE IS
        0.627 +/-0.051 VDC $

 708000 FILL,
           'TST NMBR', 'UPR LMT', 'LWR LMT', 'TST
POINT',
            C'708010',   0.678,     0.576,     C'A1-8' $

    10 PERFORM, 'VLT PROBE' $

    20 IF ,'MSRMNT' UL 'UPR LMT' LL 'LWR LMT',
THEN $

    30      ELSE $

    40      GO TO, STEP 714000 $

    50 END, IF $


C       ***********************************
        * TEST A2-8 USING ANALOG PROBE    *
        ***********************************$

C       TEST PASSES IF MEASURED VOLTAGE IS
        0.329 +/-0.051 VDC $

 709000 FILL,
           'TST NMBR', 'UPR LMT', 'LWR LMT', 'TST
POINT',
            C'709010',  0.380,     0.278,      C'A2-8' $

    10 PERFORM, 'VLT PROBE' $

    20 IF ,'MSRMNT' UL 'UPR LMT' LL 'LWR LMT',
THEN $

    30 ELSE $

    40      FILL, 'TYPE', 'DEFECTIVE PRI',
'DEFECTIVE SEC',
                2,  C'A2-6, A2-5, A2-4',  C'A4-3' $

    50      PERFORM, 'DEFECT MSG' $

    60      GO TO, STEP 999000 $

    70 END, IF $


C       ******************************
        * TEST TP1 USING ANALOG PROBE *
        ****************************** $

C       TEST PASSES IF MEASURED VOLTAGE IS
        4.310 +/-0.247 VDC $

 710000 FILL,
           'TST NMBR', 'UPR LMT', 'LWR LMT', 'TST
POINT',
            C'710010',  4.557,     4.063,      C'TP1' $

    10 PERFORM, 'VLT PROBE' $

    20 IF ,'MSRMNT' UL 'UPR LMT' LL 'LWR LMT',
THEN $

    30 ELSE $

    40      FILL, 'TYPE', 'DEFECTIVE PRI',
'DEFECTIVE SEC',
                2,   C'A3-6',  C'C3, R13-8, R13-7, R49' $

    50      PERFORM, 'DEFECT MSG' $

    60      GO TO, STEP 999000 $

    70 END, IF $


C       ***********************************
        * TEST TP2 USING ANALOG PROBE    *
        ***********************************$

C       TEST PASSES IF MEASURED VOLTAGE IS
```

```
      -3.354 +/-0.247 VDC $

 711000 FILL,
          'TST NMBR', 'UPR LMT', 'LWR LMT', 'TST
POINT',
          C'711010',  -3.107,    -3.601,     C'TP2' $

    10 PERFORM, 'VLT PROBE' $

    20 IF ,'MSRMNT' UL 'UPR LMT' LL 'LWR LMT',
THEN $

    30 ELSE $

    40      FILL, 'TYPE', 'DEFECTIVE PRI',
'DEFECTIVE SEC',
                2,      C'A4-6',     C'C4, R13-7' $

    50      PERFORM, 'DEFECT MSG' $

    60      GO TO, STEP 999000 $

    70 END, IF $


C       **********************************
       * TEST A5-3 USING ANALOG PROBE    *
       **********************************$

C      TEST PASSES IF MEASURED VOLTAGE IS
       2.155 +/-0.130 VDC $

 712000 FILL,
          'TST NMBR', 'UPR LMT', 'LWR LMT', 'TST
POINT',
          C'712010',  2.285,     2.025,     C'A5-3' $

    10 PERFORM, 'VLT PROBE' $

    20 IF ,'MSRMNT' UL 'UPR LMT' LL 'LWR LMT',
THEN $

    30      FILL, 'TYPE', 'DEFECTIVE PRI',
'DEFECTIVE SEC',
                2,      C'A5-6',    C'C5, R13-3, R13-6' $

    40      PERFORM, 'DEFECT MSG' $

    50 ELSE $

    60      FILL, 'TYPE', 'DEFECTIVE PRI',
'DEFECTIVE SEC',
                2,      C'A5-3',     C'R13-4, R13-5' $

    70      PERFORM, 'DEFECT MSG' $

    80 END, IF $

    90 GO TO, STEP 999000 $

… Note: Other Code Mixed in Here That Not Relative to
This Diagnostic Path

714000 FILL,
          'TST NMBR', 'UPR LMT', 'LWR LMT', 'TST
POINT',
          C'714010',  0.648,     0.606,     C'A1-6' $

    10 PERFORM, 'VLT PROBE' $

    20 IF ,'MSRMNT' UL 'UPR LMT' LL 'LWR LMT',
THEN $

    30      FILL, 'TYPE', 'DEFECTIVE PRI',
                1,      C'A1-6' $

    40      PERFORM, 'DEFECT MSG' $

    50 ELSE $

    60      FILL, 'TYPE', 'DEFECTIVE PRI',
'DEFECTIVE SEC',
                2,  C'A3-3, A1-4,5,9,10,12',   C'R5-2'$

    70      PERFORM, 'DEFECT MSG' $

    80 END, IF $

    90 GO TO, STEP 999000 $
```

The following is the Go-Chain of the 091200 CCA. On the first failure encountered, execution control is passed to the Diagnostician via the Atlas procedure “Diag_loop” which is shown below. This example shows that all of the Diagnostic branching logic is replaced by a simple “While-loop” structure in the test program. Even the technically untrained viewer, we believe, can easily see the increased simplicity of the test program.

Go Chain transfers control on first failure

```
B $
 111000 PERFORM, 'R1' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 112000 PERFORM, 'R2' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 113000 PERFORM, 'R3' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 114000 PERFORM, 'R4' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 115000 PERFORM, 'VDC_SIGNAL' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 116000 PERFORM, 'TEMP CAL OUTPUT' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 117000 PERFORM, 'TEMP OFFSET' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 118000 PERFORM, 'BRDG CAL' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 119000 PERFORM, 'SWOT OUTPUT' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 120000 PERFORM, 'SWOT O/P P1-B3' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 121000 PERFORM, 'SWOT O/P P1-B18' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 122000 PERFORM, 'SWOT O/P P1-B17' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 123000 PERFORM, 'SWOT O/P P1-A9' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 124000 PERFORM, 'SWOT O/P P1-B9' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 125000 PERFORM, 'SWOT O/P P1-B4' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 126000 PERFORM, 'SWOT O/P P1-B2' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 127000 PERFORM, 'SWOT O/P P1-B20' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 128000 PERFORM, 'SWOT O/P P1-B6' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 129000 PERFORM, 'SWOT O/P P1-B8' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 130000 PERFORM, 'SWOT O/P P1-A20' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
```

```
B $
 131000 PERFORM, 'SWOTP1-A20 400HZ' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 132000 PERFORM, 'SWOTP1-B1+15VDC' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 133000 PERFORM, 'SWOTP1-B1 400HZ' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 134000 PERFORM, 'SWOT P1-B14' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 135000 PERFORM, 'SWOTP1-A14' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 136000 PERFORM, 'SWOTP1-B19' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 137000 PERFORM, 'SWOTP1-A19' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 138000 PERFORM, 'BROT 1N1 INP' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 139000 PERFORM, 'BROT1N2' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 140000 PERFORM, 'BROT1N3' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 141000 PERFORM, 'BROT1N4' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 142000 PERFORM, 'BROT1N5' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 143000 PERFORM, 'BROT1N6' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 144000 PERFORM, 'BROT1N7' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 145000 PERFORM, 'BROT1N2-b' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 146000 PERFORM, 'BROT 1N1' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 147000 PERFORM, 'TEMPOT 1N3' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 148000 PERFORM, 'TEMPOT 1N1' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 149000 PERFORM, 'TEMPOT 1N2' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 150000 PERFORM, 'TEMPOT 1N4' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 151000 PERFORM, 'TEMPOT 1N5' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 152000 PERFORM, 'TEMPOT 1N6' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 153000 PERFORM, 'TEMPOT 1N7' $
      IF, 'G_PASS' EQ FALSE, THEN $
           GO TO, STEP 500000 $
      END, IF $
B $
 154000 PERFORM, 'TEMPOT 1N8' $
      IF, 'G_PASS' EQ FALSE, THEN $
```

```
            GO TO, STEP 500000 $
        END, IF $
B $
 155000 PERFORM, 'TEMPOT1N1-b' $
        IF, 'G_PASS' EQ FALSE, THEN $
            GO TO, STEP 500000 $
        END, IF $

C* END GO-NOGO CHAIN - PASSED ALL TESTS
FROM ENTRY POINT ****$
       PERFORM, 'Pass_Message' $
       GO TO, STEP 999990 $

C** FAIL - IF WE JUMP TO HERE ONE OF THE GO-
NOGO TEST FAILED *$
B$
 500000 IF, 'G_USE_DIAG' EQ TRUE, THEN $
                    PERFORM, 'Diag_loop' $
       END, IF $
       PERFORM, 'Fail_Message' $
B$
 600000 PERFORM, 'END_STATUS' $
       IF,  'G_REPEAT_TEST' EQ TRUE, THEN $
          GO TO, STEP 100000 $
       END, IF $

B$
 999990 FINISH $
 999999 TERMINATE, ATLAS PROGRAM $
```

Step 3 – Perform failure analysis and map tests

The process of test coverage mapping includes defining all tests and all measurements, and indicating the coverage of those tests and measurements across circuit failure locations and failure modes. The result of this process is a Diagnostic Knowledge Base (DKB) that is then used in run-time to dynamically, in real-time, isolate faults based upon the Diagnostician's interpretation of test results.

The Diagnostic Profiler development tool provides a number of tools and utilities that help with the task of performing UUT failure analysis. Also, the Profiler provides the overall framework for "mapping" test coverage information into the fault/symptom matrix. Using the "Specify Tests" tool, the user defines tests and measurements, and maps the coverage of tests across UUT components and failure modes. A sample is shown in Figure 3.

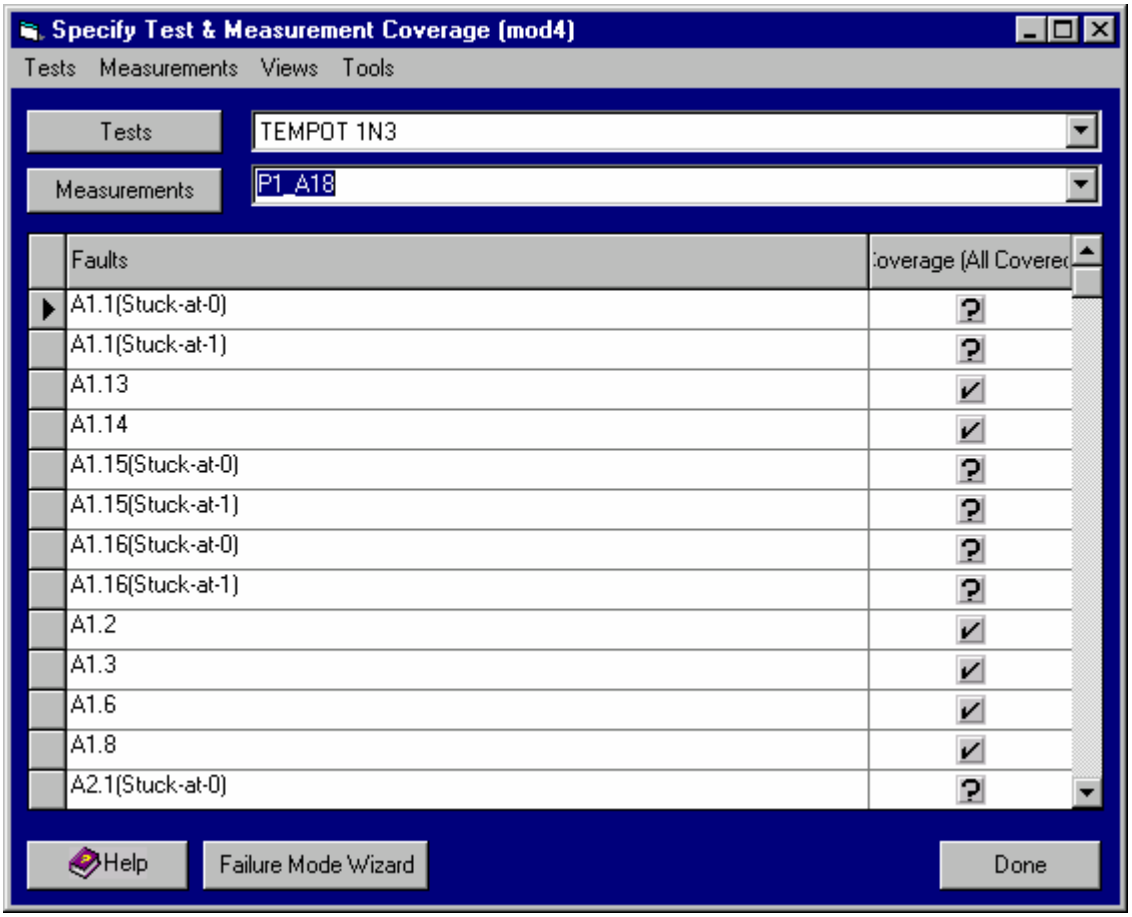


Figure 3 – Specify Tests and Measurement Tool used to Map Tests

The figure above shows a test named TEMPOT 1N3, and a measurement associated with that test named P1_A18. The test name related to the function being tested (temperature) and the measurement name relates to the circuit measurement location (P1_A18, indicating that the measurement is being made at the P1 connector, pin A18).

The list of faults shown indicates that these fault locations and fault modes are covered by that measurement. A check in the box indicates that this fault is always covered by that measurement. A question mark in the box (?) indicates that the corresponding fault or failure mode is sometimes covered by that measurement.

The Diagnostic Profiler provides numerous other tools and utilities for tailoring the diagnostic knowledge base to the test environment. These are covered in great detail in the User Manual. For further information, please contact Giordano Automation.

A part of this process involves performing testability analysis to determine how well the set of tests and measurements currently mapped “cover” the overall unit under test in terms of fault detection, fault isolation and fault resolution. Figure 4 below shows the testability analysis tool display, indicating the percentage of fault detected, percentages of faults isolated down to each possible number of parts and repair items, and the identification and composition of all ambiguity groups. Detailed reports can also be output for printing.

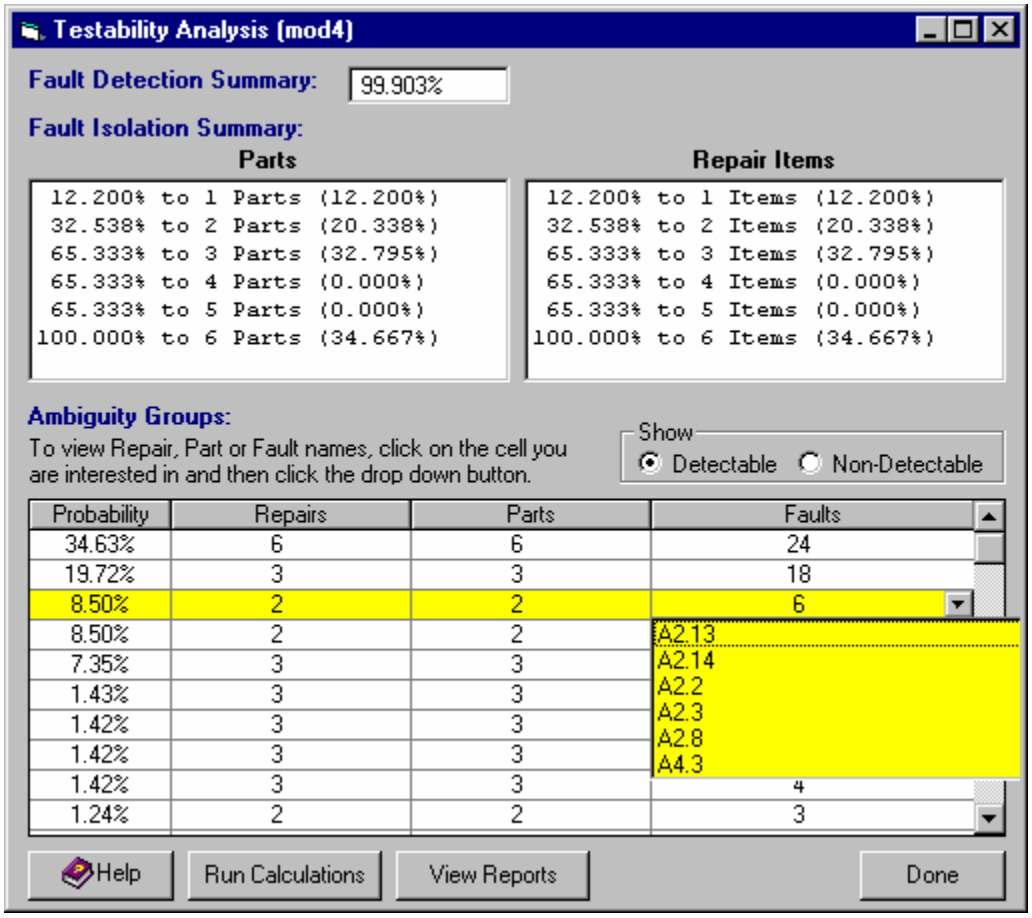


Figure 4 – Testability Analysis Display

Step 4 – Validation and Verification of the resultant Diagnostic Knowledge Base

Once the engineer is satisfied with the mapping of the tests, and that the tests result in the required level of fault isolation coverage, he proceeds to create a run-time Diagnostic Knowledge Base (DKB). This is a simple process performed by a tool in the Diagnostic Profiler. The run-time DKB is a binary file that is significantly compressed for efficient memory usage in run-time. A process within the generation of the DKB is the pre-calculation of set coverage for use in very efficient run-time operations for use with the minimum set covering algorithms used by the Diagnostician.

After the run-time DKB is created, there are a number of tools that enable validation and verification of the DKB. This V&V can be done off-line to the test station, thus freeing up valuable test station resources.

The Diagnostics V&V tool is shown in Figure 5. This tool can be used with simulated test data created by associated tools, by the engineer, or by actual test data created (and logged) on the test station.

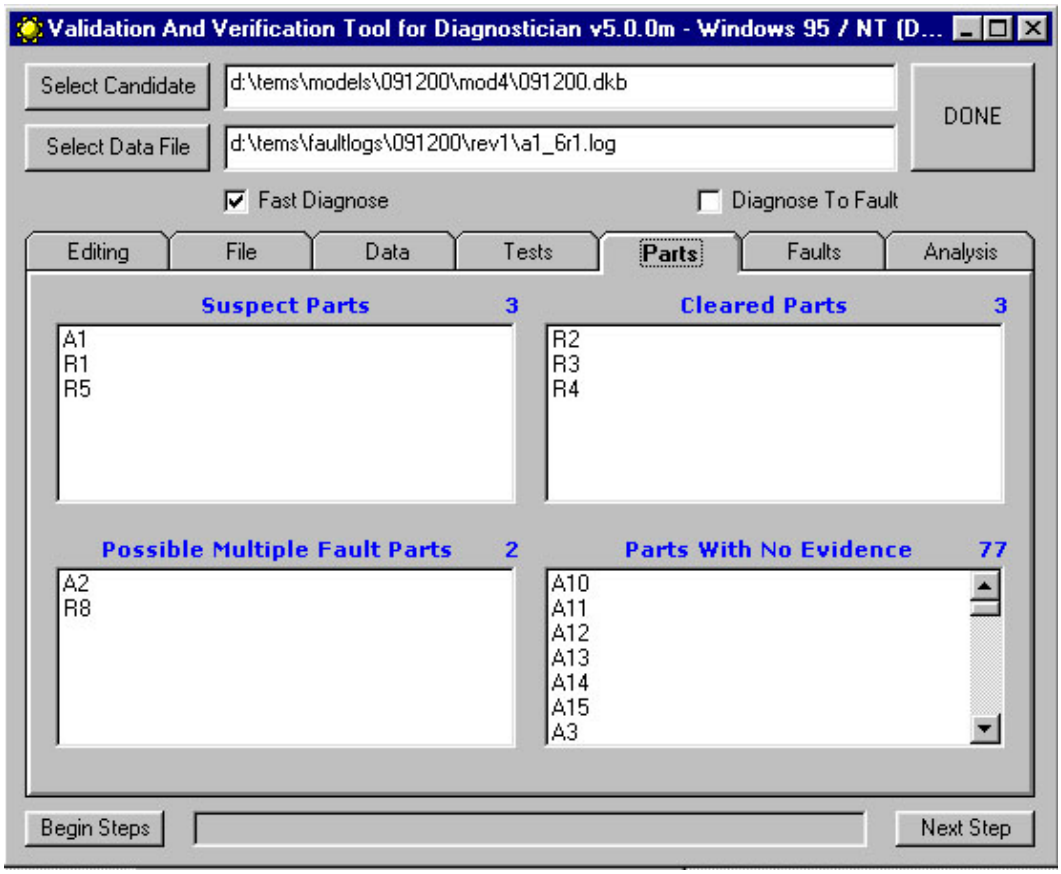


Figure 5 – Diagnostics Validation and Verification Tool

The process of using the Diagnostic Profiler's V&V tool ensures that the diagnostic behavior of the combined DKB and Diagnostician, in run-time, will exactly match that behavior of that combination on the test program. The diagnostics are thus uncoupled from the test program. By separating the tests from the diagnostic logic, overall test program V&V is significantly simplified.

Step 5 - Integrate the TPS with the diagnostic model

The fifth step in the process is to integrate the test program code with the diagnostic model. To do this, the test program code is modified to omit all diagnostic logic. The diagnostic logic is replaced with a simple "While" loop. In English, the "While" loop performs the following function: While the current ambiguity group is more than one, and there are more tests to be performed, ask the Diagnostician to identify the most significant test (the one which will most significantly resolve the fault and reduce the ambiguity group); perform that test; send the results to the Diagnostician for analysis; determine the resultant ambiguity group; return to the top of the While loop.

All the fault tree logic is replaced in the new TPS with this simple while loop that will perform the control of diagnostic test and produce an accurate callout of both single faults and possible multiple-fault conditions. This While loop is shown below.

```
023000 DEFINE,'Diag_loop', PROCEDURE $

        DECLARE, MSGCHAR, STORE, 'test-Name', 20 CHAR $
        DECLARE, INTEGER, STORE, 'status' $
        DECLARE, INTEGER, STORE, 'idx' $

        OUTPUT,
           Running Diagnostician $

        FILL, 'G_IN_DIAG', TRUE $
C****Call to Diagnostician to Retrieve Next Best Diagnostic test to Perform $
        PERFORM, 'NextTest', 'status','test-Name' $
        WHILE, 'status' EQ 0, THEN $
C****Call to Run the Next Test $
              PERFORM, 'TEST_PARSER', 'test-Name', 'status' $
              IF, 'status' NE 0 , THEN $
                    OUTPUT, Test Parser Error for ('test-Name')
                                - Returned ('status') $
                                LEAVE, WHILE $
             END, IF $
C*****Stay in the loop until there are no more diagnostically significant tests to run $
              PERFORM, 'NextTest', 'status','test-Name' $
         END, WHILE $

C***** Call to Diagnostician to Retrieve Call Out and display and log results $
         PERFORM, 'Callout', 'status' $
         PERFORM, 'TermDiag', 'status' $

     END,'Diag_loop'  $
```

Figure 6 – Diagnostician WHILE Loop

By omitting the GO-TO logic from the test program, and integrating the While loop, the test program has been restructured to make use of the Diagnostician in run-time, where the Diagnostician works in a conversational mode with the test program to perform fast, accurate and efficient diagnostic reasoning.

Step 6 - Verify the integrated TPS on MATE 390 and prepare a data package

Next, in preparation for TPS sell-off, the integrated TPS, consisting of the ATLAS test code and the DKB, is verified on the test station. This involves hosting the ATLAS code and DKB onto the station and running them together to determine appropriate operations. This also involves running fault insertions and preparation of a data package in preparation for a formal Government sell-off.

Step 7 – Perform Sell-off

A formal sell-off process was performed for each test program. The sell-off included review of the data package and fault insertions to determine that the test program operated correctly. Faults were inserted on

the UUT, and the test program was run to determine that the fault was correctly detected and isolated by the test program. An interesting note is that the Government certification team came to have a full understanding of the tools used in the overall process, and came to understand that the Diagnostic Profiler tools created a "representation" of the UUT and its diagnostic behavior. The diagnostic approach is a deterministic approach, not a probabilistic approach. Once the certification team really understood how the tools worked, and that the tools resulted in very consistent test program results, the requirement for fault insertions was reduced, with more reliance with the Profiler's V&V tools. Using the V&V tools, a much broader scope of faults can be verified than with limited fault insertion testing.

See Section 3.3 for additional details on the TPS Certification process.

Step 8 - Provide updated CPIN software on suitable media for release and distribution.

The updated CPINs were released on appropriate media and for storage in the Software Control Center, in accordance with Air Force requirements.

Step 9 - Update TPI (Test Program Instruction) documentation

The Test Program Instructions (TPI) were updated. These updates included significant overall improvements for operation of the TPS by test station operators. Additionally, corrected and up-to-date schematics, (the result of step 1) were incorporated into the TPI.

More information of the TPI content can be found in Section 3.4.

3.3 Certification of TEMS SRU Test Programs

Certification of the TEMS test programs was conducted in the course of this project. The LY Software Staff in WRALC conducted the certification in conformance to their acceptance requirements for each individual TPS. As part of the Certification process, the Air Force ran each SRU test program on the test station to verify its operation. This included both end-to-end (functional) tests as well as diagnostic tests. Representative faults were injected (simulated) in the units under test to force diagnostic test procedures to be executed. Full data logging was done during the test program execution and the logged results were printed out, and put into storage with the unit under test.

3.4 Development of Test Program Instructions (TPIs) for Designated UUT TPSs

Test Program Instructions for the TEMS SRU test programs were prepared and delivered in accordance with the requirements provided by the cognizant WRALC certification team. Giordano Automation prepared Test Program Instruction documents for the TEMS Shop Replaceable Unit (SRU) Test Program Sets (TPS) as listed in Appendix B.

As part of this task, Giordano Automation provided the Air Force with concise documentation relating to the all of the information required to operate and maintain the test program sets. In addition, much of the technical data that had been previously lost or that was previously incomplete in the various related Air Force Tech Orders was supplemented with corrected and complete information.

Some of the highlights of the improved documentation and information in the TPI are listed below:

- Inclusion of digital pictures representing Interface Test Adapter (ITA) installation and UUT setup were included.
- Full probe point listing and probing diagrams for each probe point called out in the test program.
- UUT Schematic
- UUT parts list
- UUT assembly drawings
- ITA data base
- Test Program usage of test station resources (stimulus and measurement instrumentation)
- Correlation of UUT name, LRU, designation, Part Number, Revision Level, CPIN, TO Number, Unix Directory, etc.

The cover sheets of each of the Test Program Instruction prepared for each Unit Under Test is included in Appendix B. The Test Program Instructions are delivered under a separate CDRL.

The Test Program Instructions were developed based upon the requirements specified in applicable Mil Standards and specific WRALC format requirements. The content of the Test Program Instructions include:

- Set-Up Procedure.
- List all cables required
- List ITAs required
- Diagram of the on-line set-up including the relative positioning of UUT, ITA and ATE.
- Testing Procedure: Provide program start procedures.
- Testing data table: Provide all necessary operator instructions and diagrams that are impractical to include on a test station display.

The TPI provides information needed for testing (e.g., hook-up, probe point locations, or other programmed operator intervention) which the ATE under control of the test program cannot conveniently provide. Appropriate contents are largely dependent on the ATE being used. Since graphics are not available under the MATE operating system, the pertinent data is provided in the TPI. A complete table of contents is provided. The table of contents contains the following information:

a. A listing of paragraph headings and corresponding page numbers, entitled "Contents".

b. A complete listing of figures (by figure number and title) and corresponding page numbers, entitled "List of Illustrations".

4.0 RELATED CONTRACT TASKS

4.1 SRU Test and LRU Test Correlation

In the course of the project, the SRU testing in relation to the to the A-10 IATS LRU Test and the I-ABIT LRU Test was investigated. The three TEMS EPU LRUs tested are the A10 TEMS EPU, the KC-135 Master EPU and the KC-135 Slave EPU. Test results and specific discrepancies of the LRU testing of the three EPU LRUs system to agree with the corresponding Circuit Card Assembly (CCA) testing on the MATE 390 test system. Currently, the IABIT tester is in the process of an upgrade and is not available for correlation studies. Correlation studies and SRU improvement are currently being addressed specific to

the several cards that experience difficulties at the system level, specifically the A/D converter card (A8), and the Vibration card (A5). Improvements have been made in the programs with respect to accuracies and calibration techniques that have dramatically improved the correlation.

4.2 Test/Demonstration of the TPS Process.

During the course of Integration and certification, the appropriate production and Software personnel are constantly updated and provided on-site demonstrations of the various functional operations of the Programs. During the formal certification Process, both Production and Software development personnel are not only present but actively perform the fault insertion and certification process. In addition, numerous technical meetings and demonstrations have been carried out with the specific Air Force groups including production, Software, Engineering, and the Program Manager's office of the A10/ KC135 sustainability.

4.3 Familiarization and Demonstration.

Familiarization of the modified systems as specified in the contract schedule is conducted routinely in the sell-off process. The demonstration or sell off occurs currently at WRALC in Warner Robins, Georgia.

4.4 Automated Vertical Testability.

The intention is to implement an automated vertical testability tracking system between the IA-BIT tester and the MATE 390 tester. This would include a network between the MATE 390 and the I-ABIT testers. The network hardware is available at the Mate 390. The I-ABIT tester is currently not operational to work this network. A software log that automatically records test results by TEMS item serial number, identifies any inconsistency of test results, and provides other engineering information on the cause of the inconsistency has been defined and developed for the TEMS depot test environment.

4.5 Storage of pertinent test data in a data base

Storage of pertinent test data in a data log is currently implemented on the tester.

4.6 Evaluate Air Force Research Lab (AFRL) Information Directorate Software

Giordano Automation investigated some of the software tools activities at the Rome Research Site of the AFRL, including WIN-WIN and ORBIT, and others to determine their applicability for use specifically on this project, and generally for Air Force depot maintenance support. One of the software packages that shows promise in this application is the Model Integrated Computing (MIC) Environment developed by Vanderbilt. The use of these tools becomes more applicable as we create a data management and network environment to correlate data from one level to the next and will be studied in more detail in the next phases of this project.

4.7 Software.

All computer software developed and modified (including Test Program Sets) has been delivered directly to the Government as specified in the contract schedule. All software developed under this effort has been delivered on media compatible with the target computer system, and has been properly designated using the Air Force CPIN designation system. Refer to Appendix A.

All computer software was developed using appropriate programming languages as defined by the government. Justification for the language selected is based upon system interface, interoperability, communications functions, human interface, and requirements for security, safety, and reliability. The software design makes use of existing software and for subsequent reuse to the maximum feasible extent.

Complete software documentation is provided which includes installation, user and maintenance instructions.

4.8 Progress Reports

Monthly Program Progress Reports were submitted under this specific contract number, entitled Application of Model-Based Reasoning Tools Used to Enhance and Improve Diagnostic Performance to Improve Air Force Maintenance. The progress reports were organized as follows:

1.0 Progress During the Reporting Period
2.0 Plans for the Next Reporting Period
3.0 Current Problem Areas
4.0 Estimate of Completion

4.9 Revisions to Existing Documents

This deliverable includes modifications to Test Program Instructions for each of the test programs. Due to the voluminous nature of these documents and the procedures internal to the Air Force, this data has been delivered on a CD-ROM directly to the TEMS equipment specialist. Hard and soft copies of these documents have been provided to the cognizant Air Force personnel at Warner-Robins ALC. Additionally, updated TO numbers and updated CPIN revision levels have been prepared and submitted to the Item Manager. The cover pages of the delivered T.O's are provided in Appendix B as a reference.

4.10 Test/Demonstration Plan

Each of the revised test programs has been formally certified and sold off to the cognizant Air Force personnel at Warner-Robins ALC via fault insertion demonstrations with the accompanying documentation and sign-off for acceptance. The demonstration procedures and signatory requirements were documented as an Acceptance Test Plan in coordination with the Air Force at WR-ALC.

4.11 Software Documentation (Installation, User and Maintenance Instructions)

A Diagnostic Profiler and Diagnostician User Manual and Installation CD-ROM is provided under this CDRL item.

4.12 Final Technical Report (A007)

The final technical report is provided herein. This report includes technical work accomplished, and info gathered, pertinent observations, nature of problems, positive and negative results, design criteria established, procedures followed, baseline data used, technology demonstrated, process developed, lessons learned, potential improvements, and follow-on work. There are additional TPSs that are being developed and delivered under a follow-on contract that encompasses additional TEMS circuit card test program sets.

APPENDIX A

A Copy of the Computer Software Configuration Item (CSCI) Components Delivery Forms are provided here in Attachment A

CPIN	*SRU*	*TEMS EPU Slot Configuration*
85E-USQSS/M390-U013-00A	**091150**	**A2**
85E-USQ85/M390-U004-00A	**091200-301,302**	**A3**
85E-USQ85/M390-U005-00A	**091250-302**	**A4**
85E-USQ85/M390-U006-00A	**091300-303,302**	**A5**
85E-USQ85/M390-U007-00A	**091350-302,304,305,306**	**A6 (not yet sold off)**
85E-USQS5/M390-U014-00A	**091450-(301-314)**	**A8**
85E-USQ85/M390-U014-00A	***091 460-(301-306)**	**A8**
85E-USQ85/M390-U024-00A	**9383755-10**	**A8**
85E-USQ85/M390-U008-00A	**091600 –301 thru –308, 311 thru -318, 322, 323, 325, 326**	**A11**
85E-USQ85/M390-U009-00A	**091650-303,304 (six configurations)**	**A10, A12**

EPU 091450/091460 CONFIGURATION SLOT A8
Delivery of Computer Software Configuration Item (CSCI) Components

CSCI Components The following configuration items were delivered to the LYSRP Software Control Center (SCC) on 13 Mar 2001 :

CPIN #	Revision	CPIN Date	Qty	Security Classification	Type Media
85E-USQ85/M390-U014-00A	004	16 Feb 01	2	Unclassified	3 1/2 Floppy
85E-USQ85/M390-U014-00D	004	16 Feb 01	2	Unclassified	3 1/2 Floppy

Request the following information be provided by the Weapon System Software Manager and returned to the originator either by electronic transmission or FAX (6-1316).

Distribution

	Is being accomplished by the development activity
X	Must be accomplished by the SCC – Users **are** on official ID
	Is not required

TCTO Announcement In accordance with TO 00-5-15, this software release will be announced by the method indicated below.

	*TCTO #_____
	*Letter of transmittal
	Electronic message
	Electronic bulletin board
X	Not required – Software is for Depot use only

*Announcement documents **will / will not** be provided for packaging with the software

Media Reproduction

Reproduction Equipment Nomenclature	Location of Equipment and POC
MATE 390 Tester	Bldg. 645/ POC-Ignacio Quintanilla

Approval *I certify to the best of my knowledge the above listed CSCI data is correct and acceptable. The software, having satisfactorily completed weapon system program testing, is authorized for use as CPIN masters and reproducibles for distribution. The LYSRP SCC is authorized to provide software support utilizing directions furnished on this form.*

Rohn O. Ussery LEADA/6881 X705 — 13 Mar 2001

Name, Title, Office, Phone — **Signature, Date**

EPU 9383755 CONFIGURATION SLOT A8 (Re-engineered Card)

Delivery of Computer Software Configuration Item (CSCI) Components

CSCI Components The following configuration items were delivered to the LYSRP Software Control Center (SCC) on 31 May 01 :

CPIN #	Revision	CPIN Date	Qty	Security Classification	Type Media
85E-USQ85/M390-U024-00A	000	21 Mar 01	2	Unclassified	3 ½ Floppy
85E-USQ85/M390-U024-00D	000	21 Mar 01	2	Unclassified	3 ½ Floppy

Request the following information be provided by the Weapon System Software Manager and returned to the originator either by electronic transmission or FAX (6-1316).

Distribution

	Is being accomplished by the development activity
X	Must be accomplished by the SCC – Users **are** on official ID
	Is not required

TCTO Announcement In accordance with TO 00-5-15, this software release will be announced by the method indicated below.

	*TCTO #_____
	*Letter of transmittal
	Electronic message
	Electronic bulletin board
X	Not required – Software is for Depot use only

*Announcement documents **will / will not** be provided for packaging with the software

Media Reproduction

Reproduction Equipment Nomenclature	Location of Equipment and POC
MATE 390 Tester	Bldg. 645/POC-John Hill

Approval *I certify to the best of my knowledge the above listed CSCI data is correct and acceptable. The software, having satisfactorily completed weapon system program testing, is authorized for use as CPIN masters and reproducibles for distribution. The LYSRP SCC is authorized to provide software support utilizing directions furnished on this form.*

Rohn O. Ussery LEADA/6881 X705 — **Name, Title, Office, Phone**

31 May 01 — **Signature, Date**

EPU 091300 CONFIGURATION SLOT A5
Delivery of Computer Software Configuration Item (CSCI) Components

CSCI Components The following configuration items were delivered to the LYSRP Software Control Center (SCC) on _2 May 2001_____:

CPIN #	Revision	CPIN Date	Qty	Security Classification	Type Media
85E-USQ85/M390-U006-00A	001	10 Apr 01	2	Unclassified	31/2 Disk
85E-USQ85/M390-U006-00D	001	10 Apr 01	2	Unclassified	31/2 Disk

Request the following information be provided by the Weapon System Software Manager and returned to the originator either by electronic transmission or FAX (6-1316).

Distribution

	Is being accomplished by the development activity
X	Must be accomplished by the SCC – Users **are** on official ID
	Is not required

TCTO Announcement In accordance with TO 00-5-15, this software release will be announced by the method indicated below.

	*TCTO #_____
	*Letter of transmittal
X	Electronic message
	Electronic bulletin board
	Not required – Software is for Robins Depot use only

*Announcement documents **will / will not** be provided for packaging with the software

Media Reproduction

Reproduction Equipment Nomenclature	Location of Equipment and POC
MATE 390	Bldg. 645, POC – John Hill

Approval *I certify to the best of my knowledge the above listed CSCI data is correct and acceptable. The software, having satisfactorily completed weapon system program testing, is authorized for use as CPIN masters and reproducibles for distribution. The LYSRP SCC is authorized to provide software support utilizing directions furnished on this form.*

Rohn Ussery ES/LEADA 468-6881 x 705
Name, Title, Office, Phone **signed by Rohn Ussery 9 May 01**

EPU 091200 CONFIGURATION SLOT A3
Delivery of Computer Software Configuration Item (CSCI) Components

CSCI Components The following configuration items were delivered to the LYSRP Software Control Center (SCC) on 12 Jun 2001 :

CPIN #	Revision	CPIN Date	Qty	Security Classification	Type Media
85E-USQ85/M390-U004-00A	003	7 May 01	2	Unclassified	3 ½" Floppy
85E-USQ85/M390-U004-00D	003	7 May 01	2	Unclassified	3 ½" Floppy

Request the following information be provided by the Weapon System Software Manager and returned to the originator either by electronic transmission or FAX (6-1316).

Distribution

	Is being accomplished by the development activity
X	Must be accomplished by the SCC – Users **are** on official ID
	Is not required

TCTO Announcement In accordance with TO 00-5-15, this software release will be announced by the method indicated below.

	*TCTO #_____
	*Letter of transmittal
	Electronic message
	Electronic bulletin board
X	Not required – Software is for Depot use only

*Announcement documents **will / will not** be provided for packaging with the software

Media Reproduction

Reproduction Equipment Nomenclature	Location of Equipment and POC
MATE 390 Tester	Bldg. 645 POC-John Hill/6-5303

Approval *I certify to the best of my knowledge the above listed CSCI data is correct and acceptable. The software, having satisfactorily completed weapon system program testing, is authorized for use as CPIN masters and reproducibles for distribution. The LYSRP SCC is authorized to provide software support utilizing directions furnished on this form.*

Rohn Ussery LEADA 6881 x705 — **Name, Title, Office, Phone**

12 Jun 2001 — **Signature, Date**

EPU 091250 CONFIGURATION SLOT A4
Delivery of Computer Software Configuration Item (CSCI) Components

CSCI Components

The following configuration items were delivered to the LYSRP Software Control Center (SCC) on 14 June 01 :

CPIN #	Revision	CPIN Date	Qty	Security Classification	Type Media
85E-USQ85/M390-U005-00A	002	31 May 01	2	Unclassified	3 ½" Floppy
85E-USQ85/M390-U005-00D	002	31 May 01	2	Unclassified	3 ½" Floppy

Request the following information be provided by the Weapon System Software Manager and returned to the originator either by electronic transmission or FAX (6-1316).

Distribution

	Is being accomplished by the development activity
x	Must be accomplished by the SCC – Users **are** / on official ID
	Is not required

TCTO Announcement

In accordance with TO 00-5-15, this software release will be announced by the method indicated below.

	*TCTO #_____
	*Letter of transmittal
x	Electronic message
	Electronic bulletin board
	Not required – Software is for Depot use only

*Announcement documents **will / will not** be provided for packaging with the software

Media Reproduction

Reproduction Equipment Nomenclature	Location of Equipment and POC
MATE 390 Tester	BLDG. 645/POC-John Hill

Approval

I certify to the best of my knowledge the above listed CSCI data is correct and acceptable. The software, having satisfactorily completed weapon system program testing, is authorized for use as CPIN masters and reproducibles for distribution. The LYSRP SCC is authorized to provide software support utilizing directions furnished on this form.

Rohn Ussery LEADA 6881 x705 — **Name, Title, Office, Phone**

14 Jun 01 — **Signature, Date**

EPU 091650 CONFIGURATION SLOTS A10, A12
Delivery of Computer Software Configuration Item (CSCI) Components

CSCI Components The following configuration items were delivered to the LYSRP Software Control Center (SCC) on 7 August 01 :

CPIN #	Revision	CPIN Date	Qty	Security Classification	Type Media
85E-USQ85/M390-U009-00A	002	31 July 01	2	Unclassified	3 ½ Floppy
85E-USQ85/M390-U009-00D	002	31 July 01	2	Unclassified	3 ½ Floppy

Request the following information be provided by the Weapon System Software Manager and returned to the originator either by electronic transmission or FAX (6-1316).

Distribution

	Is being accomplished by the development activity
x	Must be accomplished by the SCC – Users **are** on official ID
	Is not required

TCTO Announcement In accordance with TO 00-5-15, this software release will be announced by the method indicated below.

	*TCTO #_____
	*Letter of transmittal
x	Electronic message
	Electronic bulletin board
	Not required – Software is for Depot use only

*Announcement documents **will** be provided for packaging with the software

Media Reproduction

Reproduction Equipment Nomenclature	Location of Equipment and POC
MATE 390 TESTER	BLDG. 645/POC – John Hill

Approval *I certify to the best of my knowledge the above listed CSCI data is correct and acceptable. The software, having satisfactorily completed weapon system program testing, is authorized for use as CPIN masters and reproducibles for distribution. The LYSRP SCC is authorized to provide software support utilizing directions furnished on this form.*

Rohn Ussery/ES-LEADA- 6881 x705 — **Name, Title, Office, Phone**

7 August 01 — **Signature, Date**

EPU 091150 CONFIGURATION SLOT A2

Delivery of Computer Software Configuration Item (CSCI) Components

CSCI Components The following configuration items were delivered to the LYSRP Software Control Center (SCC) on 13 Nov 2001 :

CPIN #	Revision	CPIN Date	Qty	Security Classification	Type Media
85E-USQ85/M390-U013-00A	003	31 Oct 01	3	Unclassified	3 ½ Floppy
85E-USQ85/M390-U013-00D	003	31 Oct 01	3	Unclassified	3 ½ Floppy

Request the following information be provided by the Weapon System Software Manager and returned to the originator either by electronic transmission or FAX (6-1316).

Distribution

	Is being accomplished by the development activity
X	Must be accomplished by the SCC – Users **are** on official ID
	Is not required

TCTO Announcement In accordance with TO 00-5-15, this software release will be announced by the method indicated below.

	*TCTO #_____
	*Letter of transmittal
	Electronic message
	Electronic bulletin board
X	Not required – Software is for Depot use only

*Announcement documents **will / will not** be provided for packaging with the software

Media Reproduction

Reproduction Equipment Nomenclature	Location of Equipment and POC
MATE 390 Tester	Bldg. 640/John Hill

Approval *I certify to the best of my knowledge the above listed CSCI data is correct and acceptable. The software, having satisfactorily completed weapon system program testing, is authorized for use as CPIN masters and reproducibles for distribution. The LYSRP SCC is authorized to provide software support utilizing directions furnished on this form.*

Steve McBee/Equipment Specialist/LEADA/6-6884 — **Name, Title, Office, Phone**

Steve McBee 13 November 2001 — **Signature, Date**

APPENDIX B

A Copy of the Cover Sheets of the delivered Test Program Instructions for each of the test programs is provided here in Attachment B for reference purposes. A CD with the delivered TPI's is provided separately.

CPIN	*SRU*	*TEMS EPU Slot Configuration*	*Tech Order #*
85E-USQSS/M390-U013-00A	091150	A2	5E18-2–8-3
85E-USQ85/M390-U004-00A	091200-301,302	A3	5E18-2–8-4
85E-USQ85/M390-U005-00A	091250-302	A4	5E18-2–8-5
85E-USQ85/M390-U006-00A	091300-303,302	A5	5E18-2–8-6
85E-USQ85/M390-U007-00A	091350-302,304,305,306	A6 (not yet delivered)	5E18-2–8-7
85E-USQS5/M390-U014-00A	091450-(301-314)	A8	5E18-2–8-18
85E-USQ85/M390-U014-00A	091 460-(301-306)	A8	5E18-2–8-18
85E-USQ85/M390-U024-00A	9383755-10	A8	5E18-2–8-19
85E-USQ85/M390-U008-00A	091600 –301 thru –308, 311 thru-318,322,323, 325, 326	A11	5E18-2–8-8
85E-USQ85/M390-U009-00A	091650-303,304 (six configurations)	A10, A12	5E18-2–8-9

Document: 5E18-2-8-3
Revision: 0001
Date: 15 October 2001

TEST PROGRAM SET INSTRUCTION

Synchro Conditioner Circuit Card
Part Number 091150-302,305

For Depot Support of the
Turbine Engine
Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU)
CCA Name	Synchro Conditioner
Part #	091150 - 302, 305
CPIN	85E-USQ85/M390-U013-00A REV 003
T.O.	5E18-2-2,-3,-4
Designation	A-2
Unix Directory	/u/tems/gac91150
Test System	M390H
ITA	TEMS 6

WRITTEN BY GIORDANO AUTOMATION CORPORATION

Document: 5E18-2-8-4
Revision: 0001
Date: 07 May 2001

TEST PROGRAM SET INSTRUCTION

Bridge, Temperature and Switch Conditioner Circuit Card

Part Number 091200 –301
Part Number 091200 –302

For Depot Support of the
Turbine Engine
Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU)
CCA Name	Bridge, Temperature and Switch Conditioner
Part #	091200 – 301, -302
CPIN	85E-USQ85/M390-U004-00A REV 003
T.O.	5E18-2-2,-3,-4
Designation	A-3
Unix Directory	/u/tems/gac91200
Test System	M390H
ITA	TEMS 4

WRITTEN BY GIORDANO AUTOMATION CORPORATION

Document: 5E18-2-8,-5
Revision: Original
Date: May 31, 2001

TEST PROGRAM SET INSTRUCTION

High Level DC Conditioner
Part Number 091250-301
Part Number 091250-302

For Depot Support of the
Turbine Engine
Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU)
CCA Name	High Level DC Conditioner
Part #	091250 –301,-302
CPIN	85E-USQ85/M390-U005-00A
T.O.	5E18-2-2,-3,-4
Designation	A-4
Unix Directory	/u/tems/gac91250
Test System	M390H
ITA	TEMS 4

WRITTEN BY GIORDANO AUTOMATION CORPORATION

Document: 5E18-2-8-6
Revision: Original
Date: April 10, 2001

TEST PROGRAM SET INSTRUCTION

Vibration Signal Conditioner Circuit Card

Part Number 091300-301
Part Number 091300-302

For Depot Support of the
Turbine Engine
Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU)
CCA Name	Vibration Signal Conditioner
Part #	091300 –301, -302
CPIN	85E-USQ85/M390-U006-00A REV 001
T.O.	5E18-2-2,-3,-4
Designation	A-5
Unix Directory	/u/tems/gac91300
Test System	M390H
ITA	TEMS 4

WRITTEN BY GIORDANO AUTOMATION CORPORATION

Document: 5E18-2-8-18
Revision: Original
Date: 20 February 2001

TEST PROGRAM SET INSTRUCTION

A-D Converter Circuit Card
Part Number 091450-301 through 314
Part Number 091460-301 through 306

For Depot Support of the
Turbine Engine
Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU)
CCA Name	A-D Converter
Part #	091450 –301 through 314 091460 –301 through 306
CPIN	85E-USQ85/M390-U014-00A REV004
T.O.	5E18-2-2,-3,-4
Designation	A-8
Unix Directory	/u/tems/gac91460
Test System	M390H
ITA	TEMS 6

WRITTEN BY GIORDANO AUTOMATION CORPORATION

Document: 5E18-2-8-19
Revision: ORIG
Date: 21 March 2001

TEST PROGRAM SET INSTRUCTION

A-D Converter Circuit Card

Part Number 9383755-10

For Depot Support of the
Turbine Engine Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU)
CCA Name	A-D Converter
Part #	9383755-10
CPIN	85E-USQ85/M390-U024-00A
T.O.	5E18-2-2,-3,-4
Designation	A-8
Unix Directory	/u/tems/gac9383755
Test System	M390H
ITA	TEMS 6

WRITTEN BY GIORDANO AUTOMATION CORPORATION

Document: 5E18-2-8-8
Revision: Original
Date: **22** August 2001

TEST PROGRAM SET INSTRUCTION

RAM Circuit Card

Part Number 091600

Test of 091600 CCA Revision Levels
091600 (RAM 1) –318, -308
091600 (RAM 2) –314, 311, -304, -301
091600 (RAM 3) –317, -307
091600 (RAM 4) -315, -312, -305, -302, -325, -322, -326, -323, -316, -313, -306, -303

For Depot Support of the
Turbine Engine
Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU) TEMS Diagnostic Display Unit (DDU)
CCA Name	RAM Circuit Card
Part #	091600
CPIN	85E-USQ85/M390-U008-00A REV 002
T.O.	5E18-2-2,-3,-4
Designation	A-11
Unix Directory	/u/tems/gac91600
Test System	M390H
ITA	TEMS 4

WRITTEN BY GIORDANO AUTOMATION CORPORATION

Document: 5E18-2-8-9
Revision: 0001
Date: 12 July 2001

TEST PROGRAM SET INSTRUCTION

PROM Circuit Card
Part Number 091650-303, 304

For Depot Support of the
Turbine Engine
Monitoring System
on the
MATE 390 Test System

UUT SUMMARY	
LRU	TEMS Electronic Processor Unit (EPU)
CCA Name	PROM Circuit Card
Part #	091650 – 303, 304
CPIN	85E-USQ85/M390-U009-00A
T.O.	5E18-2-2,-3,-4
Designation	A-10, A-12
Unix Directory	/u/tems/gac91650
Test System	M390H
ITA	TEMS 4

WRITTEN BY GIORDANO AUTOMATION CORPORATION

APPENDIX C

Giordano Automation has developed an exciting and very powerful set of tools that implement model-based diagnostic reasoning. The run-time tool, Diagnostician, provides automated diagnostics and can be seamlessly integrated into any test environment. The development tool, the Diagnostic Profiler, assists the engineer in developing the run-time diagnostic knowledge base. Together, the implementation of these tools can save significant time and money in the development of a diagnostic capability, and result in more efficient diagnostics.

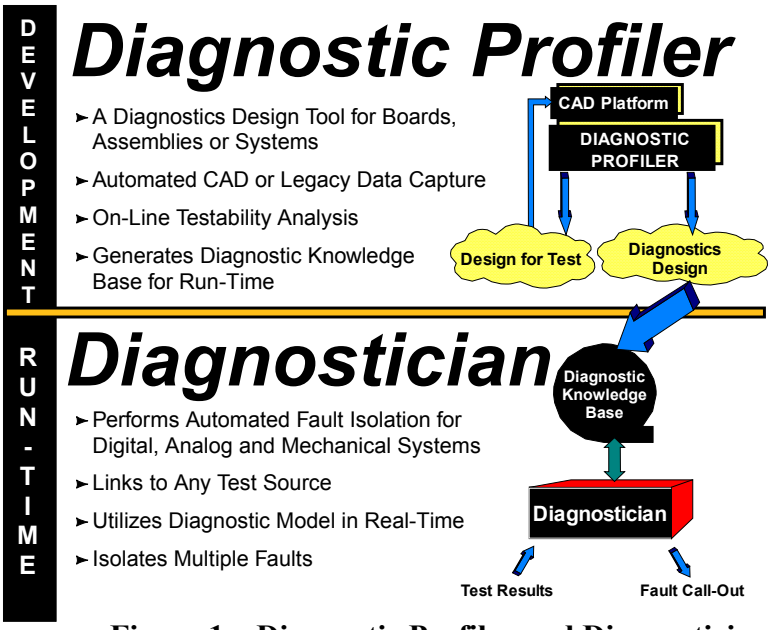


Figure 1 – Diagnostic Profiler and Diagnostician

The Diagnostician is an implementation of model-based reasoning. Model-based reasoning means that a diagnostic model of a system or item, derived from design data, serves as the basis for diagnostic reasoning. The diagnostic model is independent of the test program and independent of the sequence of tests that are run.

In the new paradigm, a model-based diagnostic software object called a Diagnostician is used in lieu of programmed fault trees. In run-time, the Diagnostician provides dynamic fault isolation without complex diagnostic logic paths, by reading test results. The diagnostic logic is not "fixed" to a pre-determined, static diagnostic tree, but rather is dynamic. The Diagnostician dynamically interprets test results - test results can come *from any source, in any order,* and with *as many or as few test results at a time* as the test source can provide. Static test trees, on the other hand, are based upon one test result at a time, in a pre-determined sequence, and from a fixed test source.

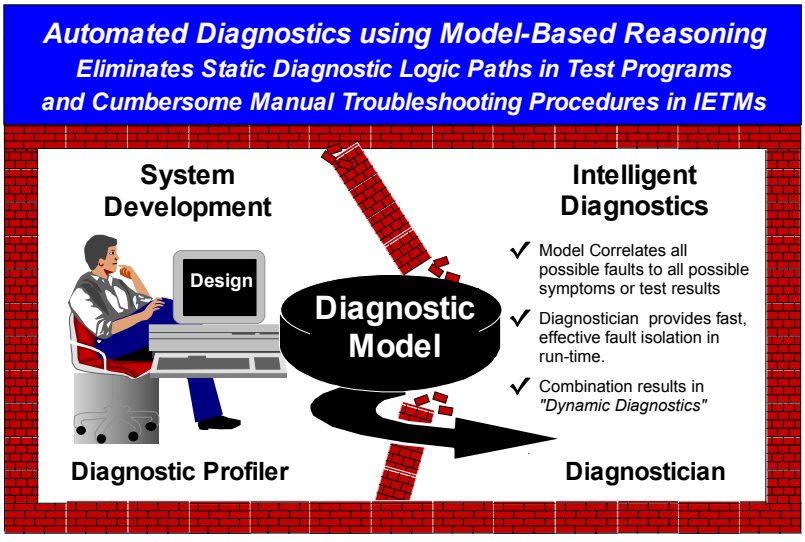


Figure 2

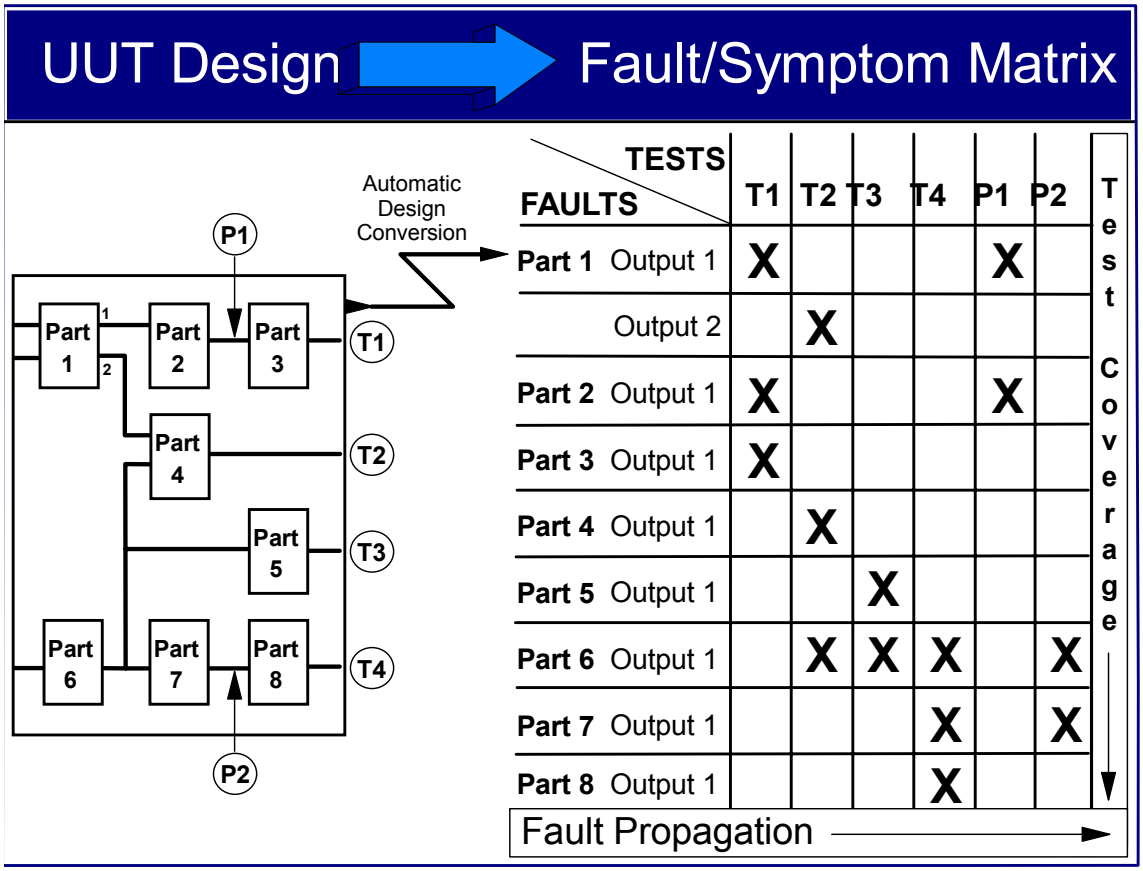


Figure 3 – Fault/Symptom Matrix Generated from Design

The Diagnostician contains a diagnostic model of the item automatically converted from design data. The model is in the form of a connectivity matrix that represents the propagation of faults (rows in the matrix) to observable measurement locations and the coverage of tests that Pass or Fail (columns in the matrix). When used in run-time, the software algorithms and knowledge base (matrix) operate to isolate faults without hard-coded diagnostic test sequences.

In run-time, the Diagnostician interprets, in real time, test results to perform fault isolation. The concept of object-oriented programs is taken full advantage of by dealing with the diagnostic logic as an independent entity of the test program. By separating the diagnostic logic from test, the test program becomes significantly simpler. Further, the diagnostic logic contained in the software object can be rehosted to any platform without any problem, because it is simply a binary file.

"Dynamic" Diagnostic Capability

- **Test Results can be input to the Diagnostician**
  - ✓ **in any order**
    - × *(no pre-set sequence)*
  - ✓ **from any source individually or in sequence**
    - × *operator observations, test instruments, data bus, data file, built-in test, automatic test equipment, system panels & displays, etc.*
  - ✓ **as many or as few at a time as the test source(s) can provide**
    - × *(not restricted to one-at-a-time to follow a diagnostic tree)*
    - × *zeroes-in on cause of fault(s)*
- **Diagnostician can identify multiple faults**
  - × *(Diagnostic trees follow single-fault assumption)*
- **Diagnostician will always zero in on cause of fault**
  - × *(never leaves the technician hanging)*
- **Will only request tests that have diagnostic significance**
  - × *based upon snapshot of current fault possibilities*

Figure 4 – Dynamic Diagnostics

Using the Diagnostician, the fundamental culture of diagnostics has been changed. Tests perform measurements and data collection and determine if those measurements are within acceptable ranges. The interpretation of what it means if the measurement has passed or failed is done by the Diagnostician, which dynamically, on-the-fly, interprets test information based upon all information it receives in any order.

The Diagnostician makes use of "Minimum Set Covering" algorithms that interpret the "Cones of Evidence" produced by both pass and fail test result data. These reasoning techniques provide for fast, accurate, flexible diagnostics, and can also isolate multiple faults. Static test trees, on the other hand, are limited to a "single fault assumption" and often do not work in multiple fault situations.

Diagnostician Implementation in a Test Program Set - a Software Engineering Perspective

In order to define the differences between traditional and model-based diagnostics, one must go back to the beginning of TPS programming. Test programs as we know them today are written as a series of functional end-to-end tests with measurements made at the output pins in order to assure that the system is operating correctly and ready for issue. The diagnostic portion is handled in one of two ways. The first is to go to a diagnostic program after the end-to-end tests are run, or to write a structured program where each test, upon failure, is followed by diagnostic tests to isolate the fault to the level required by the specification.

The traditional approach to the development of diagnostic programs requires a highly labor-intensive process of going through pages and pages of schematics and circuit diagrams, hypothesizing all potential failure conditions, and developing discrete test paths to ensure fault propagation. Highly skilled test engineers at a high cost perform this process. As system complexity increases, the ability to comprehend logic paths sometimes exceeds the ability of the human mind. Test programs have been written as long software routines with extensive branching and jumping. A single change in an independent test affects code throughout that program. In many cases, diagnostic tests are duplicated throughout the program. The development and maintenance of these programs is extremely difficult resulting in the high cost of test program sets and poor rehostability.

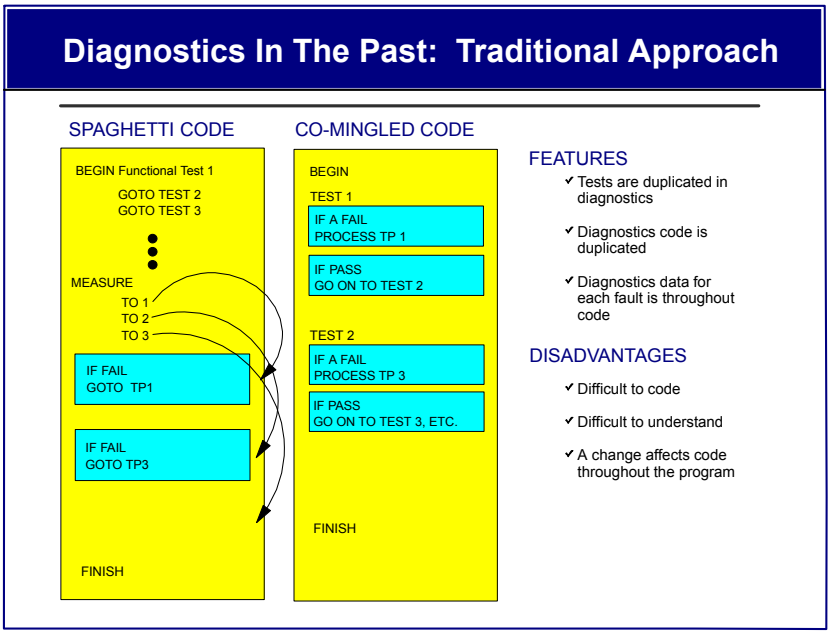


Figure 5 Traditional Test Program Structure

The technology of computer programming has evolved from unstructured code to structured code, and from structured code to object oriented code. Test programming is a special type of computer program.

As such, it too has evolved from unstructured code to structured code and will evolve into object oriented code.

In this chart, the original unstructured code is called SPAGHETTI CODE because GO-TO statements are used to control the execution flow when there are diagnostic failures. This code had the advantage of grouping all the functional tests of a good UUT together in one spot. This advantage comes from the unstructured nature of the test. This unstructured code also has two important disadvantages.

The first disadvantage is that the diagnostic routines are implicitly dependent on the functional tests run before control was transferred to them. In effect, the diagnosis is distributed between the functional tests and the diagnostic routines. In complex situations, a maintainer finds that it is difficult to pull all the data together to understand what the diagnostic routine is doing. Furthermore, any change to the functional tests, either in coverage or order, can invalidate the diagnostics routines or make them incomplete.

The other disadvantage is that the diagnostic routines contain tests that duplicate tests in the functional set of tests. The duplicated tests are selected functional tests that occur after the functional test whose failure transferred control to the diagnostic routine. Usually, this duplication is not well documented and a maintainer who changes a functional test must analyze all the diagnostic routines to carry the changes to the duplicate tests.

With the advent of structured programming, GO-TO statements were eliminated and overall program execution was made to flow in one direction. The result of applying this technology to the test program is termed CO-MINGLED CODE in the figure because the functional tests and the diagnostic routines are mingled together.

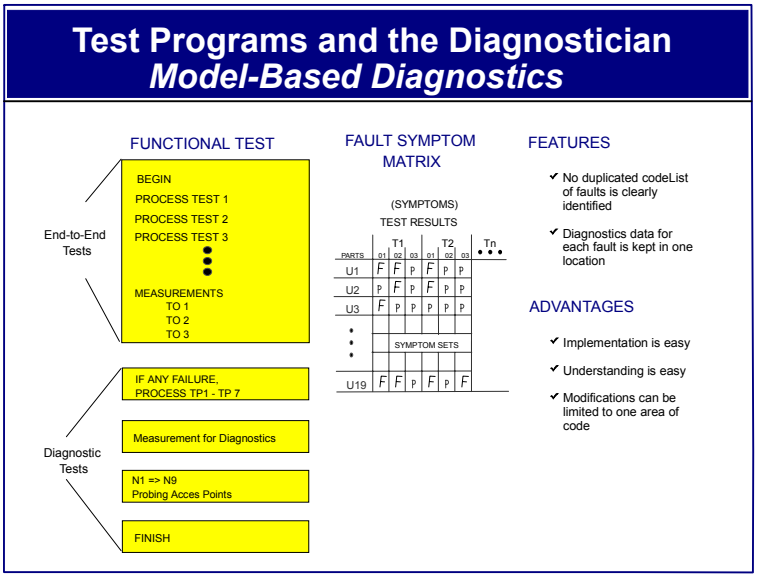


Figure 6 – Model Based Test Program Structure

The diagnostic routines of a structured test program are essentially the same as those found in the unstructured test program. Consequently, all the disadvantages of the unstructured test program apply to the structured program.

The last evolution of computer programming is to object oriented code. The basic idea is that code associated with different objects or functions is separated into units and the work gets done by the cooperation of the different units.

For test programs with diagnostics, the test (stimulus and measurements) and the diagnostic analysis are treated as separate objects. In the figure, the test objects are boxes in the left and a Fault Symptom matrix in the middle column represents the diagnostic object. The object-oriented approach is maintainable and modifiable where the earlier approaches are not.

The diagnostic information is centralized in one easy to observe Fault Symptom Matrix. In it, the relationships between tests and failures can be observed, compared to failure modes and modified. Changes in functional test order have no impact on the diagnostic process. Changes in the coverage of a test with respect to failure modes (yes/no/partial) are reflected as changes to the column of the Fault Symptom Matrix describing that test. Additions of new tests are implemented as additional Fault Symptom Matrix columns. All of these changes go to the heart of the diagnostic problem and requires no obscuring software structures.

In the object-oriented approach, duplication of tests is unnecessary. The same test can be used as part of a functional test or a diagnostic test depending on the status of the UUT being tested. The elimination of duplication greatly simplifies maintenance, reduces development cost and improves run-time effectiveness.

The result of using the Diagnostician is object oriented diagnostic capability with no Diagnostic Flow Charts.

The impact of this technology is dramatic! Savings up to 30-40% of the overall TPS costs can be realized. Maintenance of the test program, storage and use of legacy data, rehosting, updates, and porting to various platforms including portable maintenance aids are all enabled by the new paradigm. And, a Maintenance Simulator is available which allows the user to simulate the diagnostic effectiveness achieved before committing to coding the test software or building the system hardware or test hardware. Concurrent engineering of support for diagnostics is now a reality!

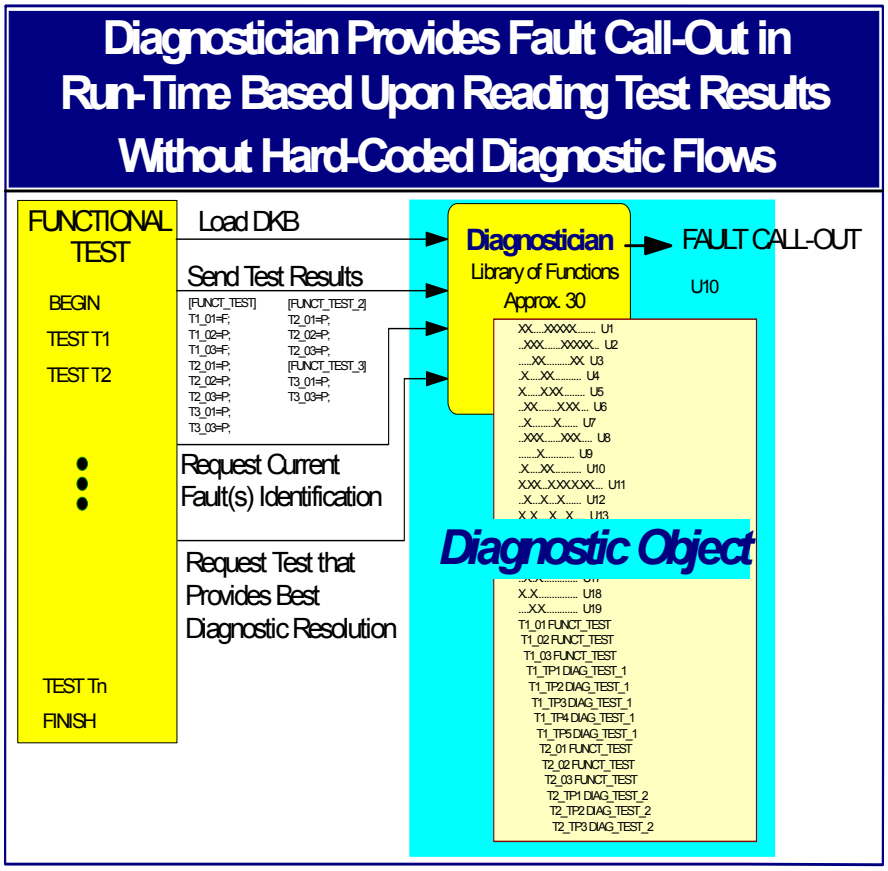


Figure 7 – Diagnostician Interaction with Test Program

The Diagnostic Profiler supports the development of the diagnostic software object (the diagnostic model). The selection of test points and the assessment of fault isolation probabilities as well as validation of these probabilities are all done using the Diagnostic Profiler during development of the TPS. Diagnostic engineering and test engineering are uncoupled. Test programming tools are used to write tests. In the process of writing these tests, the test engineer must define Pass/Fail (P/F) criteria for each response value being measured and convert test result data for each measured parameter into a P (Pass) or F (Fail). This function can be implemented utilizing a simple high level language subroutine

that accepts measurement test results and associated tolerances values as inputs and outputs a "P/F" character.

Use of the diagnostic object in run-time to perform fault isolation is done by the Diagnostician. To incorporate diagnostics into the test program, a single "WHILE" loop can be used: WHILE there is another test that can further isolate the fault, ask the Diagnostician for the next optimum test to perform, run that test, and send test results to the Diagnostician.

The methodology described is straightforward and well within the responsibilities and expertise of a test engineer. Utilizing the Diagnostician paradigm, the test engineer focuses on what he does and knows best: testing. The specifics of diagnosis, which is a function of UUT topology and behavior, is left to automated reasoning algorithms, which are better suited than a human in resolving complex diagnostic situations.

In addition to reducing TPS development time and cost, the model-based diagnostics reasoning approach is easily updated for design changes and allows fault simulation for diagnostics V&V.

Run-Time Operational View

The Diagnostician™ is a major innovation to the overall test process. To support embedded and off-line applications, the run-time Diagnostician™ has been designed to operate in a myriad of host platforms.

The new model-based diagnostics paradigm treats the diagnostic logic as an "object" which interacts with test results to perform fault isolation. In the next generation test system, a “Client” which invokes the “Services” required by the system will replace the test executive. The test object will communicate to the Diagnostician object in the Windows Dynamic Link Library (DLL) protocol. For the purpose of this discussion on interfacing the Diagnostician in the Windows-based framework, the term *Client* will be used. Client is used here as a generic name for any Windows-based software, which communicates to the Diagnostician using DLL. Note, however, that the operating modes discussed in this paper may be extrapolated to any operating system: DOS, Unix, X-Windows, VMS, or any test environment including LabVIEW, CVI, HP-VEE, ATLAS, etc.

Since Diagnostician functions are callable as "building blocks" the programmer can implement diagnostic function in any way that fits his test program structure and test philosophy. We show in the next few paragraphs, examples of three different approaches to using Diagnostician functions to effect different test strategies. These examples represent different scenarios for test execution, sequencing and program control based upon using the Diagnostician to perform diagnostics. These examples are characterized as follows:

Diagnostician in Control Example -

Where the Diagnostician manages the flow and execution of tests.

Go/No-Go Test in Control Example -

Where the Client calls and implements a set of functional, or go/no-go tests, passes the results to the Diagnostician, and the Diagnostician subsequently takes control of the flow and execution of tests.

Mixed Control Example -

Where the control of the flow and execution of tests can be passed between the Diagnostician and the test object within the Client.

In the **GO/NOGO Control Mode,** the Client software will first execute all of the go/no-go (functional/ performance) tests. If, at the end of the program, any of the tests fail, the Client initiates the Diagnostician using a simple function call and passes to it all of the test results. Next the Client requests either an ambiguity group call-out or the next best test to be executed. This mode is good for short GO/NOGO test programs where each test does not require a large amount of setup time or long testing sequences.

Go/No-Go Control Mode

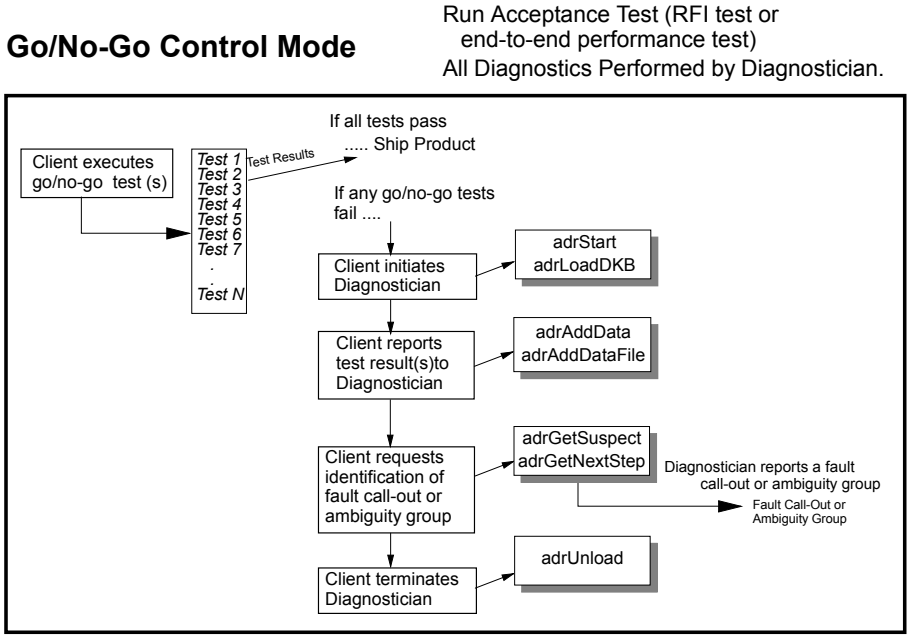


Figure 8 – Go/No-Go Control Mode

In the **Diagnostician Control Mode,** the Diagnostician is used to make all decisions on what tests are to be executed. In this mode, the Client initiates the Diagnostician before any tests are executed. Then the Client issues a DLL function call to the Diagnostician to identify the first test to be executed. The test to be executed is passed to the Client as a response to the function call. The Client will execute only those tests the Diagnostician requests until a final ambiguity group is found. The final ambiguity group is found when either the ambiguity group contains only one replaceable part, or when no more tests exist which will break up the current ambiguity group. This mode is good for tests that require a large amount of setup time or where tests are lengthy. A diagnosis can be made using the least amount of tests and testing time. Only those tests with any diagnostic significance will be executed.

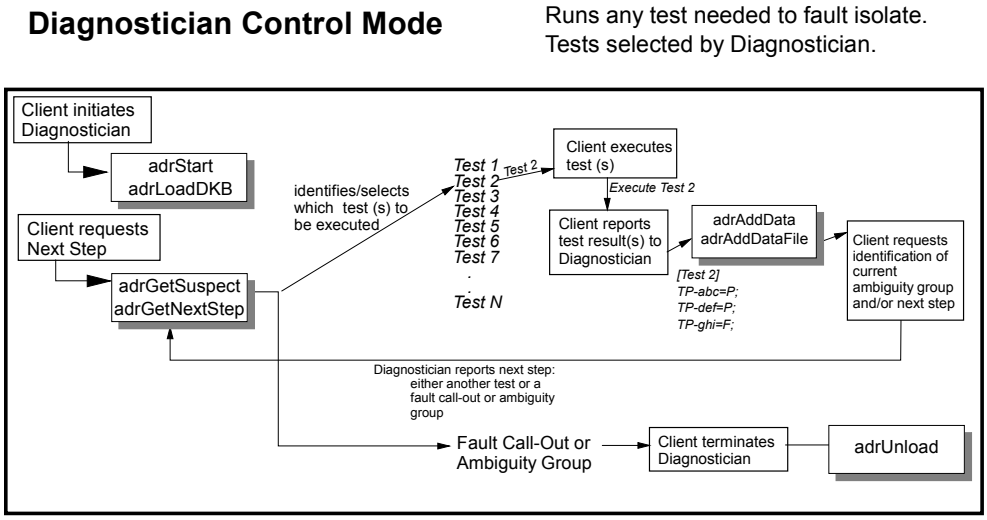


Figure 9 – Diagnostician Control Mode

The **Mixed Control Mode** is a combination of the two previous test modes. The Client will start out in the Go/No-Go Control Mode. All Go/No-Go tests will be executed and if a failure occurs, the Client will initiate the Diagnostician and perform as in the Diagnostician Control Mode. This mode can either stop at first failure in the go/no-go test or can run all go/no-go tests at once. The Mixed Control Mode is good for test programs with both short and long test sequences. The shorter tests can be executed at the top of the program. If they fail first, then the Diagnostician will reduce the number of tests and the testing time required to make a fault call-out.

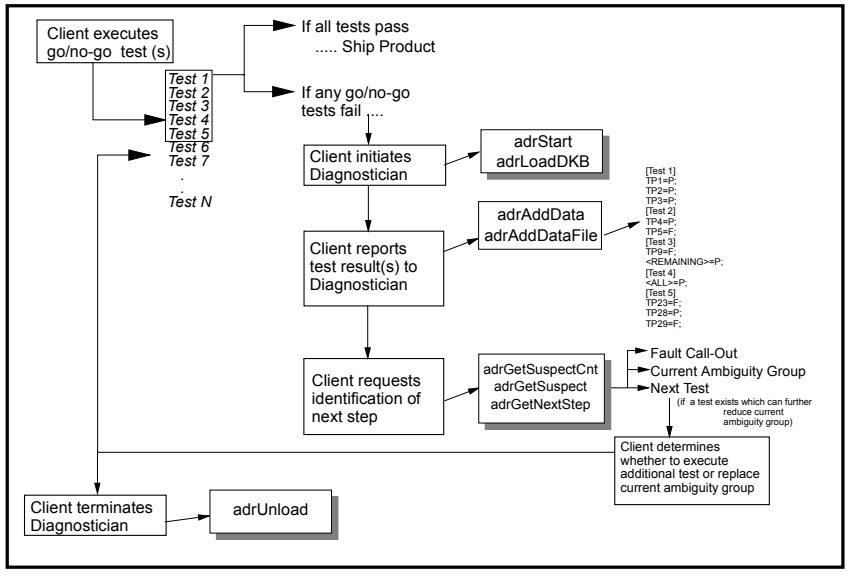


Figure 10 – Mixed Control Mode

The software architecture of the Diagnostician is that of a server. The Diagnostician provides diagnostic services to any client program. The Diagnostician acts as a server task that performs functions that provide diagnostic services. When properly interfaced on the client side, the Diagnostician functions as a library of subroutines within the client program.

The Diagnostician software, in Windows, is compiled as a Dynamic Link Library. It is a true diagnostic server that provides diagnostic services to a client program. That client program may be a test executive, test programs, LabVIEW, ATEasy, HP-VEE, or any other independent program which "sits in-between" the Diagnostician and the test program.

For example, in LabVIEW, these Diagnostician DLL function calls have been implemented as a series of virtual instruments, and the flexible test strategies in the previous discussion can be implemented easily, as shown below.

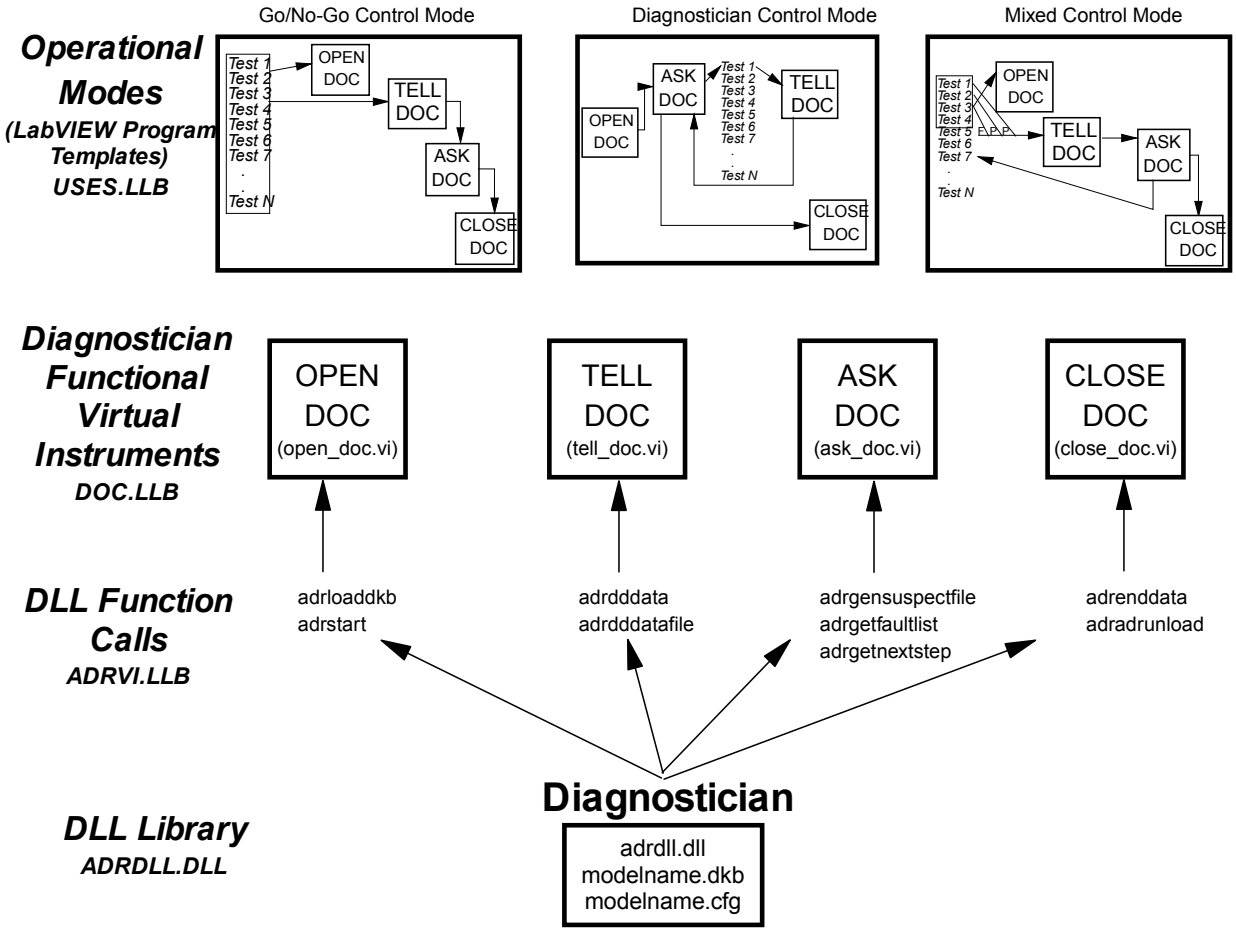


Figure 11 - Diagnostician Integration into LabVIEW Environment